An algorithm selection approach for the flexible job shop scheduling problem: choosing constraint programming solvers through machine learning⋆

David Müller[a], Marcus G. Müller[b], Dominik Kress[c,*], Erwin Pesch[a,d]

[a]*University of Siegen, Management Information Science, Kohlbettstraße 15, 57068 Siegen, Germany*
[b]*German Aerospace Center (DLR), Institute of Robotics and Mechatronics, Münchner Straße 20, 82234 Wessling, Germany*
[c]*Helmut Schmidt University - University of the Federal Armed Forces Hamburg, Business Administration, especially Procurement and Production, Friedrich-Ebert-Damm 245, 22159 Hamburg, Germany*
[d]*HHL Leipzig, Center for Advanced Studies in Management, Jahnallee 59, 04109 Leipzig, Germany*

Abstract

Constraint programming solvers are known to perform remarkably well for most scheduling problems. However, when comparing the performance of different available solvers, there is usually no clear winner over all relevant problem instances. This gives rise to the question of how to select a promising solver when knowing the concrete instance to be solved. In this article, we aim to provide first insights into this question for the flexible job shop scheduling problem. We investigate relative performance differences among five constraint programming solvers on problem instances taken from the literature as well as randomly generated problem instances. These solvers include commercial and non-commercial software and represent the state-of-the-art as identified in the relevant literature. We find that two solvers, the IBM ILOG CPLEX CP Optimizer and Google's OR-Tools, outperform alternative solvers. These two solvers show complementary strengths regarding their ability to determine provably optimal solutions within practically reasonable time limits and their ability to quickly determine high quality feasible solutions across different test instances. Hence, we leverage the resulting performance complementarity by proposing algorithm selection approaches that predict the best solver for a given problem instance based on instance features or parameters. The approaches are based on two machine learning techniques, decision trees and deep neural networks, in various variants. In a computational study, we analyze the performance of the resulting algorithm selection models and show that our approaches outperform the use of a single solver and should thus be considered as a relevant tool by decision makers in practice.

Keywords: Scheduling, Constraint programming, Algorithm selection, Machine learning, Deep neural networks

1. Introduction

Most real-world manufacturing environments are highly complex systems that feature a variety of constraints and characteristics that correspond to fairly specific settings found at the related companies. Nevertheless, they usually have similarities that can be modelled by means of generic problem formulations that reduce the complex settings to their very core. In some cases, solutions based on these problem formulations can be implemented directly. Alternatively, they may serve as building blocks or elementary subproblems in more specific optimization approaches. They are thus of major importance not only from a theoretical perspective.

Certainly, in the scheduling context, one of the most practically relevant but rather compact problem settings is the *flexible job shop scheduling problem* (FJSP). It generalizes the well-known *job shop scheduling problem*

*Corresponding author
Email addresses: david.mueller@uni-siegen.de (David Müller), marcus.mueller@dlr.de (Marcus G. Müller), dominik.kress@hsu-hh.de (Dominik Kress), erwin.pesch@uni-siegen.de (Erwin Pesch)

⋆ This is an Accepted Manuscript of an article published by Elsevier in the **European Journal of Operational Research** on 31 January 2022, available online: https://doi.org/10.1016/j.ejor.2022.01.034

(JSP; see, e.g., the survey by Błażewicz et al. 1996) and is composed of a set of jobs and a set of machines. Each job consists of a set of operations that have to be processed non-preemptively in a predefined sequence without overlapping in time in order to complete the job. Each operation is associated to a specific machine for its processing. The FJSP (see Brucker & Schlie, 1990; Blazewicz et al., 2019; Chaudhry & Khan, 2016) generalizes the latter assumption in order to take account of the fact that manufacturing systems oftentimes feature multiple machines of the same type as well as multi-purpose machines that are able to process different types of operations. More specifically, it assumes that each operation must be processed by exactly one machine out of a set of *eligible machines*. In both variants, the problem is to allocate the operations to the (eligible) machines and to sequence the operations on the machines so that some performance measure is optimized, whilst ensuring that, at any time, each machine processes at most one operation and each operation is processed on at most one machine.

Due to its relevance to scheduling theory and practice, there is a continuously growing body of literature on the FJSP, its extensions and applications. We refer the readers to Chaudhry & Khan (2016) and Blazewicz et al. (2019) for an overview. Recent research directions include the analysis of problem settings with additional practically relevant resource types as, e.g., machine operators or setup operators (e.g., Müller & Kress, 2021; Kress et al., 2019), the incorporation of sustainability aspects, e.g., energy-efficiency constraints (e.g., Rakovitis et al., 2022), or the consideration of multiple (conflicting) objectives (e.g., Türkyılmaz et al., 2022; Thenarasu et al., 2022). There is a wide variety of potential practical applications, ranging from logistics settings, e.g., in coal export terminals (Burdett et al., 2020), to machine scheduling settings in high-tech industries, e.g., in semiconductor final-test scheduling (Kress & Müller, 2022).

Since the advent of Industry 4.0, Industrial Internet of Things components allow monitoring production processes by provisioning real-time information (see Boyes et al., 2018). In order to utilize this information in operational production planning processes, the according data has to be explicitly used within the planning algorithms. Thus, in practice, manufacturing companies have to continuously compute or update feasible schedules using this data as well as other pieces of real-time data (giving rise to the fields of online scheduling or real-time scheduling, see Gershwin, 2018; Ghaleb et al., 2020). By doing so, the companies can, for example, explicitly take account of maintenance issues arising on the shop floor, delays in the arrival of raw materials, varying customer orders or due date changes. However, in order to guarantee smooth production processes without unnecessary disruptions, the underlying algorithmic approaches have to feature the ability to (re-)compute feasible schedules with an acceptable solution quality within a very short amount of time. In this regard, *constraint programming* (CP) approaches have been successfully applied to many scheduling problems and, in particular, to shop scheduling problems where disjunctive constraints are prevalent as it is the case for JSP and FJSP (Dorndorf et al., 2000, 2002). Especially, the standard CP solver provided by IBM ILOG CPLEX has shown to provide an excellent performance (see, e.g., Lunardi et al., 2020; Wari & Zhu, 2019; Laborie et al., 2018; Ham & Cakici, 2016; and, for the specific case of the FJSP, Lunardi et al., 2021; Kress & Müller, 2019). Standard CP solvers provide out-of-the-box algorithms that take the burden of understanding and implementing complex scheduling algorithms off the practitioners. They allow modeling optimization problems by making use of easy to understand variable and constraint types. Domain variables, for instance, can be used to cover temporal dimensions like the start and end of the processing of operations. In case of the FJSP, the CP Optimizer by IBM ILOG CPLEX even tends to be competitive when compared with state-of-the-art meta-heuristics (Lunardi et al., 2021; Kress & Müller, 2019). Of course, there exist alternative, non-commercial, CP solvers. One of them is included in OR-Tools, an open source software suite developed by Google. It won most of the gold medals in the 2020 MiniZinc Challenge (see MiniZinc, 2020), an annual competition that compares different CP solvers on a variety of combinatorial optimization problems (Stuckey et al., 2014). A performance evaluation between Google's OR-Tools and the IBM ILOG CPLEX CP Optimizer for the JSP was recently conducted by Da Col & Teppan (2019a) and Da Col & Teppan (2019b). The paper at hand complements this study by focussing on the FJSP. We present a thorough computational study that analyzes the relative performance of non-commercial CP solvers when compared with the CPLEX CP Optimizer for solving well-known benchmark instances taken from the literature as well as randomly generated test instances. We focus on practically relevant industry settings, where, as outlined above, the solvers must be able to quickly compute feasible solutions. To the best of our knowledge, there is only one study that compares the performance of CP solvers for the FJSP as part of the 2013 MiniZinc Challenge (MiniZinc, 2013). Note, however, that the general performance of existing CP solvers has improved rapidly over the last years (see, e.g., Laborie et al., 2018). Hence, this study, that solely considers 5 FJSP instances, is rather outdated.

In their aforementioned study, Da Col & Teppan (2019a) and Da Col & Teppan (2019b) find that, on average, CPLEX performs only slightly better than OR-Tools on small-sized problem instances of JSP. On large-scale problem instances, CPLEX outperforms OR-Tools more significantly. However, when looking at the results individually, there exist quite a few instances, where OR-Tools is able to determine better solutions or proves optimality faster than CPLEX. Similar results can be observed for many optimization problems when comparing the performance of different algorithms. Typically, there exists no single best algorithm that outperforms all other approaches on all problem instances. While a specific algorithm A may be the best overall choice for a given set of instances, another algorithm B may outperform algorithm A on a different instance set. The problem of selecting the most promising algorithm out of a given set of algorithms for a previously unseen instance of some specific problem is also referred to as the *algorithm selection problem*. It was formally introduced by Rice (1976). In order to predict the best algorithm, algorithm selection approaches generally make use of features describing the given instance. Algorithm selection approaches can be categorized into *classification methods* and *regression methods*. While the former methods directly predict an algorithm for a given instance, the latter methods predict continuous algorithm performance measures, e.g., regarding the computational time, which are then used for the selection of an algorithm.

Recently, algorithm selection approaches have been successfully applied for different problem domains. One of the most prominent examples in the field of combinatorial optimization is concerned with the propositional satisfiability problem (SAT). A corresponding algorithm selection approach, referred to as SATzilla (Xu et al., 2012, 2008), has won several medals in multiple SAT competitions. Another relevant study is presented by Kerschke et al. (2018), who analyze the complementary performance of five state-of-the-art inexact solvers for solving the traveling salesman problem on various benchmark instances and utilize this knowledge to build a very successful algorithm selector. Successful algorithm selection approaches for further concrete optimization problems are, for example, presented by Smith-Miles (2008) for the quadratic assignment problem or Wagner et al. (2018) for the traveling thief problem. Smith-Miles (2008) analyzes the relationship between characteristics of well-known problem instances and the performance of several meta-heuristics. This analysis is used to introduce algorithm selection approaches using neural networks. In Wagner et al. (2018), the authors propose various algorithm selection approaches that consider a large set of different algorithms. Their proposed algorithm selector shows an improvement when compared with the use of each single algorithm. In the domain of scheduling problems, a successful implementation of an algorithm selection approach is presented by Messelis & De Causmaecker (2014) for the multi-mode resource-constrained project scheduling problem. For building their algorithm selection approach, the authors make use of the performance complementarity of two state-of-the-art algorithms and propose regression and classification methods, the latter of which outperform the former. Other researchers are concerned with domain-independent approaches, for example in the field of automated planning (see,e.g., Rizzini et al., 2017; Cenamor et al., 2016) or for solving quantified Boolean formulas (Pulina & Tacchella, 2009). For the sake of brevity, we refer to the surveys by Kerschke et al. (2019) and Smith-Miles (2009) for detailed overviews. Besides summarizing practical algorithm selection approaches for various problem domains, these articles also cover the methodology and challenges of building an algorithm selector in general.

The success of algorithm selection approaches for the above problem domains has motivated us to develop algorithm selection approaches for the FJSP in order to leverage the performance complementarity of the CP solvers analyzed in our computational study. Our algorithm selection approaches fall into the category of classification methods. In order to predict the best solver for a given instance, we make use of two machine learning techniques. First, we make use of decision trees, which are commonly used in the abovementioned studies (see, e.g., Kerschke et al., 2018; Cenamor et al., 2016; Messelis & De Causmaecker, 2014; Pulina & Tacchella, 2009). Second, we make use of deep neural networks that have - to the best of the authors' knowledge - barely been used for implementing algorithm selection approaches (an example is presented by Smith-Miles, 2008), while they have recently proven very successful in other applications like speech or image recognition. A similar observation is due to Kraus et al. (2020), who find that the Operations Research (OR) community "is still in its infancy with regard to adopting" the deep learning technology. In an extensive literature review of papers published between October 2018 and September 2019 across the major OR journals, the authors find only three papers that explicitly make use of deep learning techniques. A directly related survey on the possibilities of leveraging machine learning to solve combinatorial optimization problems (rather than selecting from existing algorithms) is presented by (Bengio et al., 2021). An overview that focusses on reinforcement learning in scheduling settings is presented by Wang et al. (2021).

It is important to note that the set of features that is used for characterizing an instance is highly relevant for the performance of an algorithm selection approach. Therefore, we compute a diverse set of features based on instance characteristics. However, the necessary prior information that is needed for selecting appropriate features is not always available in practice. Moreover, the definition of features needs expert knowledge and their computation has to be executed in a preprocessing step. Therefore, we additionally analyze the case of providing unprocessed instance data, i.e., the processing times of operations on their eligible machines, that do not result from pre-computing specific feature values. In an extensive computational study, we analyze the performance of our algorithm selection approaches and show that they perform better than a single solver.

Our research is also related to the research fields of algorithm configuration and hyper-heuristics. Algorithm configuration refers to the problem of obtaining parameter settings of a given algorithm to optimize the performance on given problem instances (see, e.g., Hutter et al., 2014). Hyper-heuristics are approaches that select or generate heuristics for solving a given problem based on a given set of heuristics or components (see Burke et al., 2013). In the context of production scheduling, several approaches have been proposed by, e.g., Vázquez-Rodríguez & Petrovic (2010); Hart et al. (1998); Dorndorf & Pesch (1995).

Summing up, the contribution of this paper is twofold. First, we provide a computational study on the performance of commercial and non-commercial state-of-the-art CP solvers on a wide variety of benchmark instances of FJSP. Second, we develop multiple variants of an algorithm selection approach that aims to leverage the performance complementarity of the CP solvers. The overall setup is such that these approaches are applicable for practitioners in the sense that the computational times are in ranges that allow their usage in real-time scheduling approaches that require comparatively quick computations of feasible schedules.

The remainder of this paper is structured as follows. First, in Section 2, we provide a formal definition of the FJSP. The computational study on the performance of the considered CP solvers is provided in Section 3. Next, in Section 4, we develop our algorithm selection approaches in detail, and then evaluate their performance in Section 5. We close the paper with a conclusion in Section 6.

2. Definition of the flexible job shop scheduling problem

The FJSP is defined as follows. Given is a set $J = \{J_1, \dots, J_n\}$ of n *jobs*. Each job $J_i \in J$ consists of a set of q_i *operations* $O_i = \{i_1, \dots, i_{q_i}\}$ that have to be processed on a set $M = \{M_1, \dots, M_m\}$ of m *machines*. The processing of operations must not be preempted. The sets O_i are assumed to be linearly ordered for all $i \in \{1, \dots, n\}$. This relates to the fact that, for any pair of operations $i_j, i_{j'} \in O_i$ with $j < j'$, $i_{j'}$ may only start to be processed after the processing of i_j has completed. Each operation $i_j \in O_i$, $i \in \{1, \dots, n\}$, must be processed on exactly one machine out of a non-empty set of *eligible machines* $M_{i_j} \subseteq M$. Each machine can process only one operation at a time and each operation can be processed by at most one machine at a time. The processing time of an operation $i_j \in O_i$ of a job $J_i \in J$ on an eligible machine $M_k \in M_{i_j}$ is denoted by $p^k_{i_j} \in \mathbb{N}^+$. For the sake of notational convenience, we additionally define the set of *eligible operations* of a machine $M_k \in M$ as $E_k = \{i_j | J_i \in J, i_j \in O_i, M_k \in M_{i_j}\}$. We assume that all jobs and machines are available at the beginning of the planning horizon. The problem is to find a schedule, i.e., an assignment of operations to eligible machines and a sequencing of the operations on the machines, that is feasible with respect to the constraints stated above and that optimizes some performance measure. The completion time of an operation $i_j \in O_i$ of job $J_i \in J$ in a schedule is denoted by C_{i_j} and the completion time of job $J_i \in J$ is denoted by C_i. A job is completed if all of its operations are completed, i.e., $C_i = C_{i_{q_i}}$ for all $i \in \{1, \dots, n\}$. We restrict our attention to the objective of minimizing the makespan $C_{max} = \max_{i \in \{1,\dots,n\}} C_i$. The resulting problem is strongly NP-hard as it generalizes the JSP, which is well known to be NP-hard in the strong sense when aiming to minimize the makespan (Lenstra & Rinnooy Kan, 1979).

In order to facilitate the understanding of the considered problem setting, Figure 1 illustrates a feasible solution for an example instance of FJSP with three machines and four jobs by means of a Gantt chart. Jobs

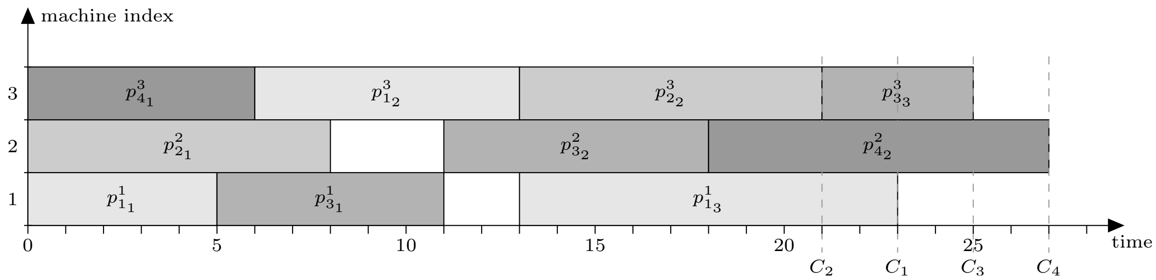


Figure 1: Exemplary illustration of a feasible solution for an instance of the FJSP

J_1 and J_3 consist of three operations, while jobs J_2 and J_4 consist of two operations. In the depicted solution, machine M_1 processes operations 1_1, 3_1, and 1_3, machine M_2 processes operations 2_1, 3_2, and 4_2, and machine M_3 processes operations 4_1, 1_2, 2_2, and 3_3.

3. Performance evaluation of constraint programming solvers

In order to assess the performance of relevant CP solvers (see Section 3.1), we conducted extensive computational tests on a variety of test instances (see Section 3.2). The tests were performed on a PC with an Intel® Core™ i7-4770 CPU, running at 3.4 GHz, with 16 GB of RAM under a 64-bit version of Windows 8. The results are presented in Section 3.3.

3.1. Constraint programming solvers

The CP solvers analyzed in our computational study are listed in Table 1. Note that CPLEX and OR-Tools are optimization suites that provide not only CP solvers. In our tests, however, we solely focussed on the respective CP solvers.

Table 1: Overview of CP solvers

Solver	Version	Modeling language	Reference
Choco	4.0.4	MiniZinc	Prud'homme et al. (2016)
Chuffed	0.10.4	MiniZinc	Chu et al. (2019)
IBM ILOG CPLEX	12.9.0	OPL	IBM (2019a)
Gecode	6.2.0	MiniZinc	Schulte et al. (2019)
Google OR-Tools	7.2	MiniZinc and directly via API	Google (2019a)

As the performance of a solver depends on the modeling of a problem, we used a solver-independent modeling language called MiniZinc. Broadly speaking, MiniZinc allows to formulate problems in a way that is close to their mathematical formulation. It allows the use of familiar notation, the definition of predicates and functions and, thus, eases the process of structuring a model by a user (Stuckey et al., 2020). The MiniZinc problem specification separates the definition of the general model and the data that defines some particular instance. To ensure readability by a solver, the MiniZinc specifications are translated to a low-level solver input language, called FlatZinc, by considering solver dependent constraint specifications. For further details, we refer to Nethercote et al. (2007); Stuckey et al. (2020). For our tests, we made use of a MiniZinc model for the FJSP that was part of the MiniZinc Challenge 2013 (Stuckey et al., 2014). This model includes *search annotations* that define search strategies for finding feasible solutions. Below, we will also consider the pure model without these annotations. The FlatZinc format is supported by all solvers listed in Table 1 except CPLEX. For the latter CP solver, we therefore used a FJSP model provided in the corresponding online documentation (IBM, 2019b) which makes use of IBM's Optimization Programming Language (OPL) and a set of modeling features that allow covering temporal dimensions that are especially relevant in scheduling problems (details are given in Laborie et al., 2018; IBM, 2019a). Google's OR-Tools provide similar features (variable and constraint types) for modeling scheduling problems (Google, 2019b), which we made use of in an additional implementation of the FJSP in Java using the OR-Tools library via its API.

In line with the requirements in real-world manufacturing systems outlined in Section 1, especially regarding their focus on real-time scheduling strategies, we set the time limit for each solver to rather small values. We considered two settings, 300 seconds and 30 seconds. Moreover, we activated the parallel processing mode in each solver.

3.2. Test instances

We considered benchmark instances from the literature as well as randomly generated test instances. All literature instances are available online via the FJSP instance collection (Mastrolilli, 2020). We restrict ourselves to summarizing the main features of the instances sets in this section. More details are summarized by Behnke & Geiger (2012) and Mastrolilli & Gambardella (2000).

- Instance set BR (Brandimarte, 1993): This set is composed of 10 (available) instances with 10 to 20 jobs, 4 to 15 machines, and 3 to 15 operations per job. The maximum number of eligible machines of each operation ranges from 2 to 6. The processing times range from 1 to 20 time units.
- Instance set CH (Chambers & Barnes, 1996): The set consists of 21 instances that were generated from three challenging JSP problem instances, introduced by Fisher & Thompson (1963) and Lawrence (1984), by "replicating" machines. The processing times of operations are identical on all of their eligible machines and correspond to the one of the original problem instance. The set features instances with 10 to 15 jobs, 11 to 18 machines, and 10 to 15 operations per job.
- Instance set DA (Dauzère-Pérès & Paulli, 1997): The set consists of 18 instances with 10 to 20 jobs, 5 to 10 machines, and 15 to 25 operations per job. The set of eligible machines of each operation was constructed randomly by assuming that each machine is eligible with a 10 to 50 percent probability. The processing times range from 10 to 100 time units.
- Instance set HU (Hurink et al., 1994; Mastrolilli, 2020): These benchmark instances are based on three JSP problem instances by Fisher & Thompson (1963) and, according to Mastrolilli & Gambardella (2000) and Behnke & Geiger (2012), 40 JSP problem instances by Lawrence (1984). The set of eligible machines of each operation consists of the associated machine of the original problem instance and, in addition, any of the other machines with a given probability. By considering different values for these probabilities, Hurink et al. (1994) generated three different instance sets, denoted by edata, rdata, and vdata. The average number of eligible machines per operation is 1.15 in edata, 2 in rdata and $m/2$ in vdata, so that the degree of flexibility is lowest in the instances of the first set. The processing times of operations are identical on all of their eligible machines and correspond to the ones of the original problem instance. According to Behnke & Geiger (2012), the remaining instances associated to Hurink et al. (1994) in the database available via Mastrolilli (2020) are based on benchmark instances by Adams et al. (1988), Carlier & Pinson (1989) and Applegate & Cook (1991) and were generated in the same manner. This results in a total of 198 HU instances.

Our random testbed, denoted by RA, is composed of three classes of instance sets with 10 jobs (small instances), 20 jobs (medium instances) or 30 to 40 jobs (large instances), respectively. Each class consists of five sets (sfjsp1–sfjsp5, mfjsp1–mfjsp5, lfjsp1–lfjsp5) with different settings regarding the remaining parameters. Each set features ten randomly generated test instances with the parameter ranges illustrated in Table 2. Hence,

Table 2: Parameters of random testbed

small instances					medium instances					large instances				
set	n	m	q_i	$\lvert M_{i_j}\rvert$	set	n	m	q_i	$\lvert M_{i_j}\rvert$	set	n	m	q_i	$\lvert M_{i_j}\rvert$
sfjsp1	10	2	$[1,3]$	$[1,2]$	mfjsp1	20	2	$[1,3]$	$[1,2]$	lfjsp1	30	3	$[2,3]$	$[1,2]$
sfjsp2	10	2	$[2,3]$	$[1,2]$	mfjsp2	20	3	$[2,3]$	$[1,2]$	lfjsp2	30	5	$[2,5]$	$[1,3]$
sfjsp3	10	3	$[1,4]$	$[1,2]$	mfjsp3	20	3	$[2,5]$	$[1,2]$	lfjsp3	40	5	$[2,5]$	$[2,3]$
sfjsp4	10	3	$[2,4]$	$[1,3]$	mfjsp4	20	4	$[5,8]$	$[2,3]$	lfjsp4	40	8	$[5,12]$	$[4,5]$
sfjsp5	10	3	$[3,5]$	$[1,3]$	mfjsp5	20	6	$[8,10]$	$[1,3]$	lfjsp5	40	10	$[8,12]$	$[4,5]$

there are 150 instances in our random testbed. While the number of jobs n and the number of machines m are fixed for the instances of each set, the number of operations per job and the number of eligible machines for each operation were drawn from uniform distributions over the intervals given in the table. Based on this data, the set of eligible machines was randomly determined for all operations. Here, each machine was selected with the same probability. The processing times were then generated as follows. First, auxiliary integer parameters p_{i_j} for all jobs $J_i \in J$ and operations $i_j \in O_i$ were drawn from uniform distributions over the interval $[10, 100]$. Based on these parameters, we constructed varying processing times over the corresponding eligible machines by drawing integer values $p_{i_j}^k$ from uniform distributions over the interval $[\lfloor(1-p)\cdot p_{i_j}\rfloor, \lfloor(1+p)\cdot p_{i_j}\rfloor]$, where $p = 0.1$. Note that, in general, the value of p influences the variation of the processing times of an operation over its eligible machines. For $p = 0$, we assume that all processing times associated to the operation are identical to the value p_{i_j}. The rather small value of 0.1 is in line with industrial settings, where the eligible machines of an operation are usually technologically similar. All test instances of the RA set are available in supplementary files that accompany this paper (Müller et al., 2021).

3.3. Computational results

Define $p_{i_j}^{\min} = \min_{M_k \in M_{i_j}} p_{i_j}^k$ for all $J_i \in J$ and $i_j \in O_i$, and set $P = \sum_{i=1}^n \sum_{i_j \in O_i} p_{i_j}^{\min}$. Then, a simple lower bound on the makespan of a given instance of FJSP is as follows:

$$LB = \max \left\{ \max_{i \in \{1,\ldots,n\}} \sum_{i_j \in O_i} p_{i_j}^{\min}, \left\lceil \frac{P}{m} \right\rceil \right\}.$$

Note that, for the sake of brevity, we do not explicitly state the concrete instance in the definition of the bound. We make use of this bound to measure the quality of a (not necessarily optimal) solution with makespan $C_{\max}$ returned by one of the considered CP solvers within the time limit with the *quality ratio* $Q = 100 \cdot (C_{\max} - LB)/LB$. In addition, we introduce a scoring system inspired by the one used in MiniZinc (2013). For a given instance, we assign one point to a solver, if it proves optimality faster or finds a better solution than all of the other solvers. If the solvers are indistinguishable, e.g., if the makespan of the two best solutions is equal without having proven optimality, no point is assigned.

The computational results over the entire set of test instances (in total 397 test instances) are presented in Table 3. For each solver and each time limit, it presents information about the percentage of instances for which

Table 3: Performance of CP solvers

	Time limit: 300 seconds					Time limit: 30 seconds				
CP Solver	feas. [%]	opt. [%]	Q_{avg}	t_{avg} $[s]$	score	feas. [%]	opt. [%]	Q_{avg}	t_{avg} $[s]$	score
Choco*	99.24	23.68	39.39	240.5	0	99.24	13.85	88.2	27.7	0
Choco	99.24	23.43	47.16	242.27	0	99.24	13.6	108.43	27.67	0
Chuffed*	100	2.02	96.22	296.15	0	100	1.01	97.18	29.81	0
Chuffed	42.82	4.53	267.9	278.39	0	27.96	1.51	280.31	28.66	0
CPLEX	100	46.85	8.17	164.94	132	100	42.32	8.44	18.83	150
Geocode*	99.24	0.25	103.33	299.37	0	99.24	0.25	103.73	30.02	0
Geocode	99.24	0.5	113.51	298.56	0	99.24	0.25	113.88	29.97	0
OR-Tools (API)	100	44.58	13.5	178.53	26	100	33.5	31.99	21.49	17
OR-Tools*	99.75	51.64	8.67	151.59	135	99.75	46.6	13.47	17.42	138
OR-Tools	89.67	49.87	10.5	140.77	42	83.38	44.33	12.14	15.8	32

a feasible or optimal solution was obtained within the given time limit (columns “feas.” and “opt.”), the average quality ratios (columns “Q_{avg}”), the average runtimes (columns “t_{avg}”), and the sum of scoring points (columns “score”). The asterisk symbol ($*$) in the solver column indicates that the underlying MiniZinc model includes the aforementioned search annotations.

It can be seen that, with the exception of Chuffed on the MiniZinc model without search annotations, the solvers are competitive with respect to their ability to determine feasible solutions. Note, however, that some solvers had conversion errors relating to the transformation of the MiniZinc specifications to the solver-input language FlatZinc for three instances of the BR set and one instance of the HU set. The search annotations specified in the MiniZinc model for the FJSP have a positive impact on the performance of the solvers (Chuffed, Gecode and OR-Tools). Moreover, it can be concluded that CPLEX provides the overall best performance with respect to the solution quality. This could, amongst other reasons, be a result of the fact that CPLEX uses the most sophisticated search strategies among the considered solvers (Laborie & Rogerie, 2016), that, e.g., integrate machine learning techniques. A more detailed analysis shows that this strength of CPLEX is particularly pronounced for instances of larger size and when the time limit is set to a lower value. However, when analyzing the percentage of instances for which an optimal solution was obtained, CPLEX is narrowly beaten by OR-Tools on MiniZinc with search annotations. This result is also reflected in the distribution of the scoring points. CPLEX and OR-Tools are the only solvers that received scoring points. For 62 (60) instances no clear winner regarding our scoring system (solvers indistinguishable) was found for a time limit of 300 (30) seconds. On this general level, these results are in line with the computational studies conducted by Da Col & Teppan (2019a) and Da Col & Teppan (2019b), which demonstrate the strengths of CPLEX especially on large-scale problem instances of JSP, but also show that OR-Tools is competitive on classical benchmark instances. With respect to the large-scale problem instances of JSP (up to 100000 operations in total), the authors observe that CPLEX performs better on instances with a relatively large number of jobs and a small number of operations per job. OR-Tools shows the opposite behavior, i.e., demonstrates a better performance on instances with a relatively small number of jobs and a large number of operations per job. Interestingly, in case of the FJSP and with respect to the solution quality, we observe that CPLEX clearly outperforms OR-Tools on instances with a relatively large average number of operations per job, as encountered in the DA instances as well as on the sets lfjsp4–lfjsp5 of the RA instances.

With respect to the different OR-Tools modes, most points were obtained when using MiniZinc with search annotations. The two other modes (API mode and MiniZinc without search annotations) obtained the majority of their points only because of determining optimal solutions faster than MiniZinc with search annotations. However, the corresponding time differences are in the range of a few milliseconds. Hence, it is reasonable to solely focus on the MiniZinc mode that is best on average. In Table 4, we therefore present the distribution of scoring points when restricting the analysis to CPLEX and OR-Tools on MiniZinc with search annotations. Apparently, OR-Tools on MiniZinc with search annotations and CPLEX tend to be competitive in terms of

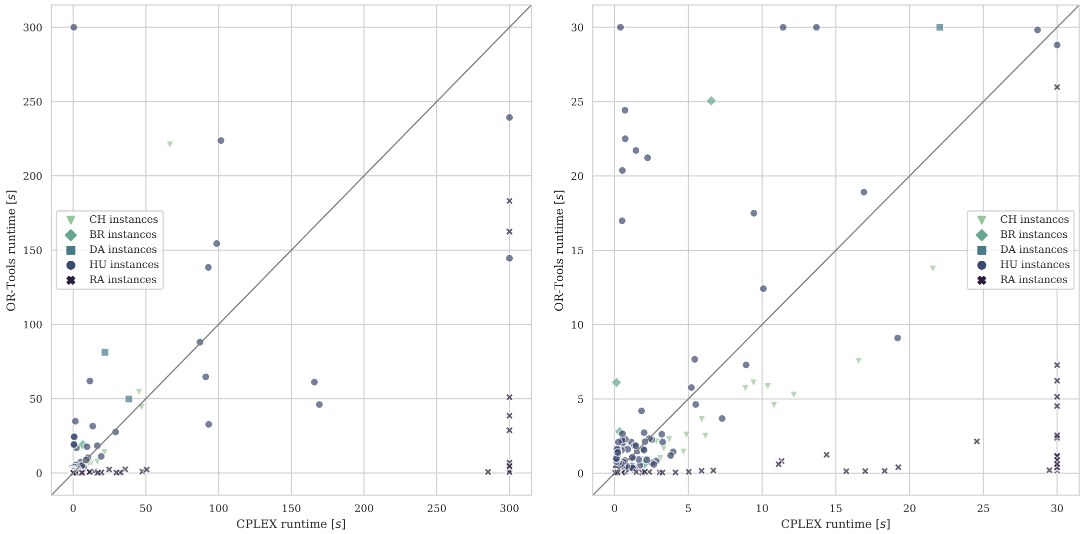
Table 4: Distribution of scoring points when restricting the analysis to CPLEX and OR-Tools*

	Time limit: 300 seconds	Time limit: 30 seconds
CPLEX	149	165
OR-Tools*	210	210

performance, i.e., one solver outperforms the other on some instances, while it is beaten by the other solver on some other instances. This can be quantified when using the *competitiveness ratio* introduced by Messelis & De Causmaecker (2014). Let T be the set of all 397 test instances. Furthermore, denote by A the set of instances for which some specific algorithm X outperforms another algorithm Y with respect to some criterion. Set B is defined analogously as the set of instances on which X is beaten by Y. Then, the competitiveness ratio c is defined by $c = 2 \cdot \min\{\frac{|A|}{|T|}, \frac{|B|}{|T|}\}$. Obviously, a ratio close to one indicates that X and Y perform similarly well.

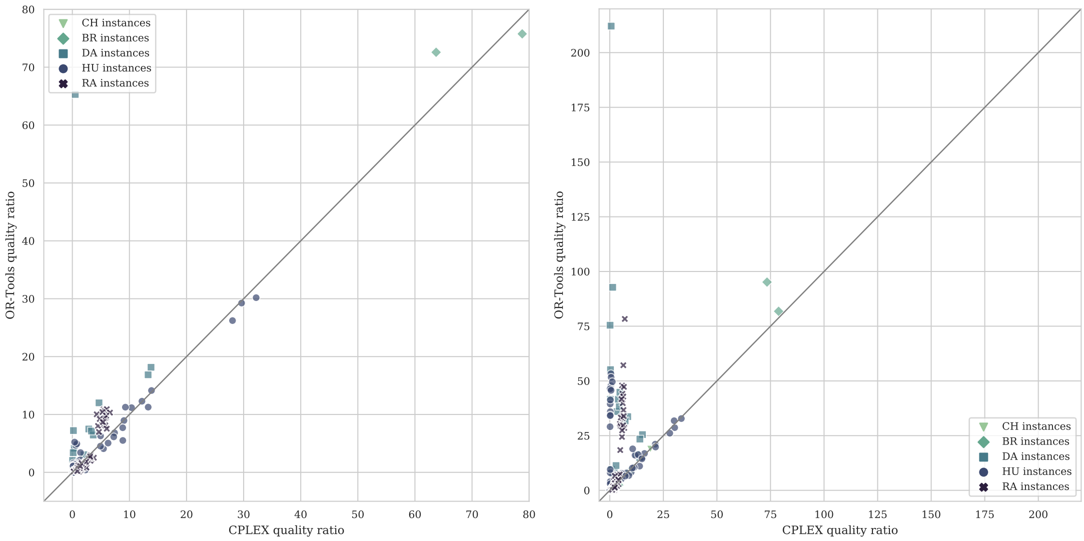
In our case, based on the scoring points presented in Table 4, the competitiveness ratio of CPLEX and OR-Tools on MiniZinc with search annotations is 0.751 for the 300 seconds time limit and 0.831 for the 30 seconds time limit. Note that, if we assigned a point to both solvers in case of a tie, these values were even larger. Hence, we will restrict our attention to these two settings in the remainder of this paper. For the sake of brevity, we will not explicitly specify the fact that we are making use of the MiniZinc search annotations and we will omit the asterisk when referring to OR-Tools.

In order to gain further insights into the competitiveness of both solvers, a more detailed analysis that takes account of the different sets of test instances is presented in Figures 2 and 3.



(a) Time limit: 300 seconds (b) Time limit: 30 seconds

Figure 2: Performance of CPLEX and OR-Tools - ability to determine provable optimal solutions



(a) Time limit: 300 seconds (b) Time limit: 30 seconds

Figure 3: Performance of CPLEX and OR-Tools - quality ratios

Figure 2 illustrates the runtimes of CPLEX and OR-Tools when restricting the attention to the instances for which at least one of the two solvers was able to determine an optimal solution, including the proof of optimality. The data is represented by means of scatter plots, with each data point representing a specific test instance.

The maximum values of the runtimes depicted in the plots represent the time limits of 300 and 30 seconds, respectively. Hence, data points at the very top or right of the plots represent instances on which one of the solvers was not able to determine an optimal solution within the time limit. If both solvers have determined an optimal solution for a specific instance within the same time, the respective data point lies on the diagonal lines in the plots. As can be seen, there exist quite a few instances that were solved to optimality by both solvers within milliseconds or a few seconds. However, for some instances, the performance of the solvers is significantly different, which is in line with the findings of Pesch & Tetzlaff (1996). As an example, we observe that (at least for the smaller time limit), CPLEX seems to perform better than OR-Tools on the HU instances, in particular on the instances with a relatively large number of machines and a relatively large average number of eligible machines per operation as prevalent in the rdata and vdata instance sets. For the RA and CH instances, on the other hand, OR-Tools seems to perform better. In case of the RA instances, this is especially apparent for the instance sets sfjsp4–sfjsp5 and mfjsp1–mfjsp2, which feature test instances with a relatively small average number of operations per job as well as a relatively small number of machines.

Figure 3 focusses on the quality ratios of the computed solutions. For reasons of clarity, we omit data points for which the quality ratio of both solvers is identical. As already observed in Table 3, CPLEX outperforms OR-Tools on most instances, especially when the time limit is set to only 30 seconds. However, on a few instances, OR-Tools was able to compute better solutions.

In summary, we conclude that there clearly is a potential payoff when using an algorithm selection approach based on the CP solvers CPLEX and OR-Tools for the FJSP. Hence, the remainder of this paper aims at developing such an approach.

4. Algorithm selection approaches

Rice (1976) introduces an abstract model for the algorithm selection problem. It consists of the following elements:

- A *problem space* X, that consists of a set of instances of a problem. In our case, the FJSP represents the (optimization) problem under consideration.
- An *algorithm space* A, that includes a set of algorithms for (potentially heuristically) solving the problem instances. Here, based on the computational results presented in Section 3, A is composed of the CP solvers CPLEX and OR-Tools with specified time limits.
- A *performance measure* y, which is used to quantify the performance of an algorithm on a given problem instance. We make use of the scoring system introduced in Section 3.3.

The algorithm selection problem is to find a mapping $S : X \to A$ that maps each problem instance to an algorithm that maximizes the performance measure y. Usually, in order to (heuristically) obtain a mapping, one defines a *feature set* F to characterize the problem instances and then, for example, applies machine learning techniques to select an appropriate algorithm based on the concrete features of an instance.

This section proceeds as follows. In Section 4.1, we define an appropriate feature set for FJSP. Hereafter, in Section 4.2, we randomly construct a problem space that aims at providing a sufficiently large amount of instances to later construct and evaluate our algorithm selection approaches based on machine learning techniques. Our procedure of generating the problem space is such that some of the instances mimic the ones of the sets introduced in Section 3.2. After a preliminary analysis on the competitiveness of the CP solvers on the problem space in Section 4.3, our algorithm selection approaches are introduced in detail in Section 4.4.

4.1. Feature set

The definition of the feature set is a crucial part of any algorithm selection approach as it is used to characterize the instances and then link these characteristics to the performance of the algorithms.

In order to be able to define a feature set in a compact manner, we introduce some additional notation. Given a data multiset X consisting of k real values $x_1, \dots, x_k$, we denote the smallest value of the multiset by $\min(X)$ and the largest value of the multiset by $\max(X)$. Furthermore, we denote the arithmetic mean

$$\frac{1}{k}\sum_{i=1}^{k} x_i$$

by $mean(X)$ and the corrected standard deviation

$$\sqrt{\frac{1}{k-1}\sum_{i=1}^{k}(x_i - mean(X))^2}$$

by $sdev(X)$. Finally, when assuming that the multiset X is sorted in the order of non-decreasing values, we denote the nine deciles of this sorted multiset by $dec_1(X), \dots, dec_9(X)$.

We define the following multisets associated to an instance of FJSP:

- $\mathcal{Q} = \{q_1, \dots, q_n\}$
- $\bar{\mathcal{Q}} = \{\frac{q_1}{mean(\mathcal{Q})}, \dots, \frac{q_n}{mean(\mathcal{Q})}\}$
- $\mathcal{M} = \{|M_{1_1}|, \dots, |M_{1_{q_1}}|, \dots, |M_{n_1}|, \dots, |M_{n_{q_n}}|\}$
- $\bar{\mathcal{M}} = \{\frac{|M_{1_1}|}{mean(\mathcal{M})}, \dots, \frac{|M_{1_{q_1}}|}{mean(\mathcal{M})}, \dots, \frac{|M_{n_1}|}{mean(\mathcal{M})}, \dots, \frac{|M_{n_{q_n}}|}{mean(\mathcal{M})}\}$
- $\mathcal{E} = \{|E_1|, \dots, |E_m|\}$
- $\bar{\mathcal{E}} = \{\frac{|E_1|}{mean(\mathcal{E})}, \dots, \frac{|E_m|}{mean(\mathcal{E})}\}$
- $\mathcal{P} = \left\{ \frac{\sum_{M_k \in M_{i_j}} p^k_{i_j}}{|M_{i_j}|} | J_i \in J, i_j \in O_i \right\}$
- $\bar{\mathcal{P}}$: Includes the elements of $\mathcal{P}$ divided by $mean(\mathcal{P})$.
- $\hat{\mathcal{P}}$: Includes one element for each job. This element is defined by the corrected standard deviation of the average processing times of the operations over their eligible machines.

According to Kerschke et al. (2019), features should be informative, interpretable, cheaply computable, generally applicable, and complementary. Following these principles, we define a total of 151 features which are grouped into four categories: instance size related (28 features), flexibility related (50 features), processing time related (37 features), and priority rule related (36 features). Three of the instance size related features directly correspond to parameters of an instance, i.e., the number of jobs n, the number of machines m, and the total number of operations $\sum_{i=1}^{n} q_i$. The other features of the first three categories are related to the multisets defined above. They are listed in Table 5. Within the table, an 'x' indicates that we make use of the according feature(s).

Table 5: Multiset-based features related to instance size, flexibility and processing time

Category	Multiset X	$\min(X)$	$\max(X)$	$mean(X)$	$sdev(X)$	$dec_1(X), \ldots, dec_9(X)$
Instance size related	$\mathcal{Q}$	x	x	x	x	x
	$\bar{\mathcal{Q}}$	x	x	-	x	x
Flexibility related	$\mathcal{M}$	x	x	x	x	x
	$\bar{\mathcal{M}}$	x	x	-	x	x
	$\mathcal{E}$	x	x	x	x	x
	$\bar{\mathcal{E}}$	x	x	-	x	x
Processing time related	$\mathcal{P}$	x	x	x	x	x
	$\bar{\mathcal{P}}$	x	x	-	x	x
	$\hat{\mathcal{P}}$	x	x	-	x	x

The priority rule related features are structurally different from the features of the three prior categories, as they rely on heuristically computing feasible schedules in order to determine the feature values (see Mirshekarian & Šormaz, 2016, for a similar approach). To do so, we apply a priority rule based approach proposed by Kress et al. (2019) that follows an idea of the list scheduling algorithm proposed by Giffler & Thompson (1960). In each iteration, this heuristic selects time instant and a machine, and appends an eligible operation that can start being processed at the respective point of time with respect to the precedence constraints at the end of the machine's operation sequence. Among all potential operations (referred to as candidate operations), exactly one operation is chosen based on a priority rule. We implemented four different priority rules, namely *shortest processing time*, *longest processing time*, *least work remaining*, and *most work remaining* (for details, see, e.g., Haupt, 1989; Dorndorf & Pesch, 1995).

For each of the four resulting schedules, we compute the following features:

- Arithmetic mean of job completion times
- Minimum, maximum, and corrected standard deviation of the job completion times divided by the corresponding arithmetic mean
- Arithmetic mean of machine loads, where the load of a machine is defined as the completion time of the operation that is scheduled last on the machine
- Minimum, maximum, and corrected standard deviation of the machine loads divided by the corresponding arithmetic mean

Moreover, during runtime of the heuristic, we count the total number of candidate operations and define the final count divided by the total number of operations as another feature.

4.2. Problem space generation

The successful development and evaluation of algorithm selection approaches depends on a sufficiently large problem space. Therefore, as the benchmark set introduced in Section 3.2 is rather small, we randomly generated a sufficiently large number of instances. The underlying procedure is identical to the one for the RA set in Section 3.2. It is used to generate instances with 10 to 40 jobs and 2 to 8 machines. In order to take account of our results in Section 3.3, the remaining parameter ranges are such that the generation procedure is likely to result in a set that includes instances in accordance with the literature sets introduced in Section 3.2. They are illustrated in Table 6. For each reasonable combination of parameter values, we generated 25 test instances. Recall, that the parameter p is used for generating the processing times based on values p_{i_j}, $i \in \{1, \ldots, n\}$,

Table 6: Parameters used for generating the problem space

n	10, 20, 30, 40
m	2, 3, 4, 5, 6, 7, 8
q_i	$[1,3]$, $[1,4]$, $[2,4]$, $[2,6]$, $[2,8]$, $[4,10]$, $[6,12]$, $[10,15]$
$\lvert M_{i_j}\rvert$	$[1,2]$, $[2,3]$, $[4,5]$
p_{i_j}	$[10,100]$
p	0, 0.1

$i_j \in O_i$. For $p = 0$, the processing times of operations are identical on all of their eligible machines, which mimics the instances of the CH set and the HU set. In total, the problem space is composed of 27200 instances.

4.3. Preliminary processing and analysis

In line with the computational results presented in Section 3.3, we first analyze the competitiveness of the considered solvers on the problem space. As we focus on practically relevant industry settings, we fix the time limit to 30 seconds.

Both solvers were able to determine a feasible solution for all instances. For 4216 instances, they perform identically with respect to the above scoring system, so that a clear winner cannot be identified. We deleted these instances from the problem space. Hence, the remainder of this paper focusses on the remaining 22984 instances. In a next step, we randomly partitioned this set of instances into a *training set* (80 %, 18388 instances) used for the training of our algorithm selection approaches and a *validation set* (20 %, 4596 instances) used for the pre-evaluation of our approaches (see Section 4.4). Afterwards, for the final validation of our approaches, our algorithm selection approaches were tested on a separate *test set* composed of previously unseen instances (see Section 5). With respect to the distribution of scoring points achieved by the solvers on the training set, we find that both solvers show a complementary performance with CPLEX receiving 9583 scoring points and OR-Tools receiving 8805 scoring points, resulting in a competitiveness ratio $c = 0.958$. In line with our results in Section 3.3, this allows to conclude that there exists a potential payoff when using algorithm selection approaches.

Figure 4 allows to take a closer look on the competitiveness of the solvers on the instances of the training set.

For exemplary features of the feature set, the bar plots illustrate the strengths and weaknesses of the solvers by representing the distribution of the scoring points achieved by each solver on the training set. As can be seen, OR-Tools tends to perform better than CPLEX when the arithmetic mean of the number of operations of jobs is rather small. This effect decreases for an increasing number of jobs or machines. Similarly, an increase of the arithmetic mean of the number of eligible machines for each operation is in favor of CPLEX, which seems to be independent of the number of jobs or machines.

Based on the plots in Figure 4 or corresponding scatter plots, it is almost impossible for the human eye to identify instance clusters that are expressive enough to allow a substantiated choice of a solver when being confronted with a previously unseen instance. We therefore make use of machine learning techniques to tackle this task. This allows to implicitly gain knowledge about a high quality mapping S.

4.4. Algorithm selection models

Since the desired output of our algorithm selection approaches, i.e., a prediction of the best solver, is categorial, our approaches fall into the category of classification methods. The instances of the problem space are labelled with binary labels according to our preliminary analysis of Section 4.3, i.e., each instance is associated with a value that indicates the solver that received the respective scoring point. We make use of two technologies: decision trees (Section 4.4.1), which are frequently used in the field of algorithm selection (see, e.g., Kerschke

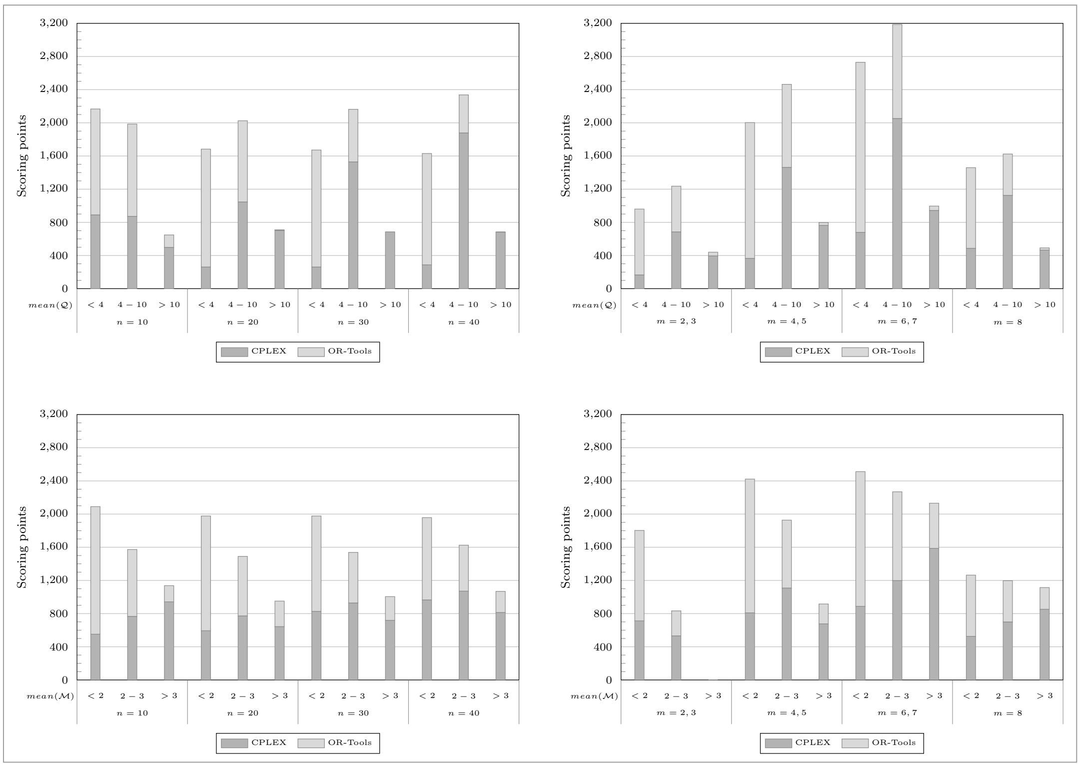


Figure 4: Bar plots illustrating the performance competitiveness between CPLEX and OR-Tools on the training set

et al., 2018; Messelis & De Causmaecker, 2014), and deep neural networks (Section 4.4.2), which are less common in the OR literature (Kraus et al., 2020). In order to prevent overfitting and in order to specify the details, e.g., the network architecture, of our models, we pre-evaluate our models on the validation set. A model's performance is measured in terms of its *accuracy*, i.e., the percentage of correct classifications on the validation set.

4.4.1. Decision trees

Decision trees can be graphically illustrated by flowchart-like tree structures, in which each node (except the leaf nodes) represents a test on a feature, each edge represents a result of the test, and each leaf node represents a final prediction. There exists a broad variety of decision tree techniques and algorithms. For the sake of brevity, instead of summarizing well-known results, we refer the readers to Han et al. (2011).

For building our decision tree models, we used the Waikato Environment for Knowledge Analysis, WEKA, in version 3.8.3 (Frank et al., 2016; Hall et al., 2009). WEKA provides a broad collection of machine learning techniques. Amongst others, it offers a variety of different decision tree techniques and algorithms that can be utilized to construct an algorithm selector:

- Decision stump (DS): method for constructing a one-level decision tree using a single feature (Iba & Langley, 1992)
- Hoeffding tree (HT): incremental decision tree algorithm (Hulten et al., 2001)
- J48: generates a C4.5 decision tree (Quinlan, 1993)
- Logistic model tree (LMT): algorithm for building decision trees with logistic regression functions at the leaves (Landwehr et al., 2005; Sumner et al., 2005)
- Random tree (RT): constructs a tree that considers a set of randomly chosen features at each node (Frank et al., 2016)

- Random forest (RF): method to construct a variety of random trees (Breiman, 2001)

- REP tree (REPT): builds a tree using information gain and reduced-error pruning (Frank et al., 2016)

We made use of these algorithms with the software's default parameters on our training set. The results of the pre-evaluation of the resulting decision tree models on the validation set are presented in Table 7. For each of the models, the table presents the accuracy. Based on these results, we decided to select the best three variants, RF, LMT, and REPT, for evaluation on the test set in Section 5.

Table 7: Pre-evaluation of decision tree models on the validation set

Decision tree algorithm/model	DS	HT	J48	LMT	RF	RT	REPT
Accuracy [%]	77.52	82.2	82.66	85.36	86.36	81.38	84.25

4.4.2. Deep neural networks

With recent advances in machine learning, deep learning has become an increasingly popular technique (Schmidhuber, 2015; Hinton et al., 2006; Bengio et al., 2007; Kraus et al., 2020). It refers to the use of deep neural networks that are inspired by neuroscience and date back to McCulloch & Pitts (1943). These networks can be represented by graphs. A respective graph is structured into multiple layers (see, e.g., Goodfellow et al., 2016; Han et al., 2011): an input layer, one or more hidden layers, and an output layer. Each layer consists of a collection of units, also referred to as neurons or nodes. These units receive input from other units, or - in case of the input layer - are fed with given data (in our case, instance feature values), compute some value as a function (typically a weighted sum) of the inputs and, based on this value, compute an output via a potentially nonlinear activation function, which is then passed on to connected units or serves as an output of the network. The number of units of the input layer therefore represents the number of elements of an input data tuple, while the units in the output layer are associated with the output values, e.g., the class labels. Usually, the layers are sequentially connected and, thus, build up a feed-forward structure. If, additionally, each unit of a layer is connected with each unit in the succeeding layer, the network is referred to as a *fully connected neural network*. After having defined a network topology, the weights of the network (as well as bias values) are computed and adjusted during a learning process on the training set, e.g., by making use of backpropagation, in order to minimize a loss function that measures the error between the predicted values and true observations.

In our case, as we solely consider the choice among two solvers, we restrict ourselves to network topologies with a single output unit that computes a real value in the range $(0,1)$. This value represents the probability that a specific solver, say OR-Tools, is the best choice for the considered instance. A value close to zero then predicts CPLEX to perform better, while a value close to one predicts OR-Tools to be the best choice. We make use of the binary cross-entropy loss function. Denote by K some set of instances of the problem space and, for some $k \in K$, let $\tilde{y}_k \in (0,1)$ be the output of the network and $y_k \in \{0,1\}$ be the binary label, representing the best solver for this instance. Then, this loss function is defined as

$$L = -\frac{1}{|K|}\sum_{k=1}^{K} y_k \log(\tilde{y}_k) + (1-y_k)\log(1-\tilde{y}_k). \tag{1}$$

For implementing our deep neural networks, we used the open source machine learning framework PyTorch in version 1.6.0 (PyTorch, 2020) using Python. In order to determine a final network architecture that does not tend to overfit and that has an acceptable accuracy, we implemented and trained several network architectures with varying parameter settings and pre-evaluated these networks on the validation set. Throughout this empirical evaluation, we found the following fully connected neural network, denoted by FCNN, to perform best (accuracy: 86.28%):

- Input layer: 151 units according to the number of features introduced in Section 4.1
- Three hidden layers with 256, 128, and 64 units using piecewise linear activation functions, in our case rectified linear unit (relu) ones (see, e.g., Goodfellow et al., 2016; Han et al., 2011),
- Output layer: one unit using a non-linear activation function, in our case a sigmoid activation function (see, e.g., Goodfellow et al., 2016; Han et al., 2011)

The learning process was performed with Adam (Kingma & Ba, 2014), a popular adaptive first-order gradient-based optimization algorithm, with default parameters as suggested by the authors. The number of epochs, specifying the number of learning phases over the entire training set, was set to 10. The batch size, i.e., the number of instances used in a learning iteration, was set to 100. Moreover, for better generalization and robustness of the learning process, we normalized the input data to values between 0 and 1.

There exist advanced network architectures that target at specific data structures (see, e.g., Goodfellow et al., 2016; Kraus et al., 2020). *Convolutional neural networks*, for example, aim to exploit spatial dependencies in grid-like data, e.g., among neighboring pixels in images. These networks provide special filtering and pooling techniques to reduce the dimension and to identify certain patterns within the data (see LeCun et al., 2010). In our case, grid-like structures arise, when providing unprocessed instance data in form of the processing times of operations on their eligible machines to a neural network. Hence, for this case, we additionally trained a convolutional neural network, denoted by CNN. It is illustrated in Figure 5. The (normalized) processing times

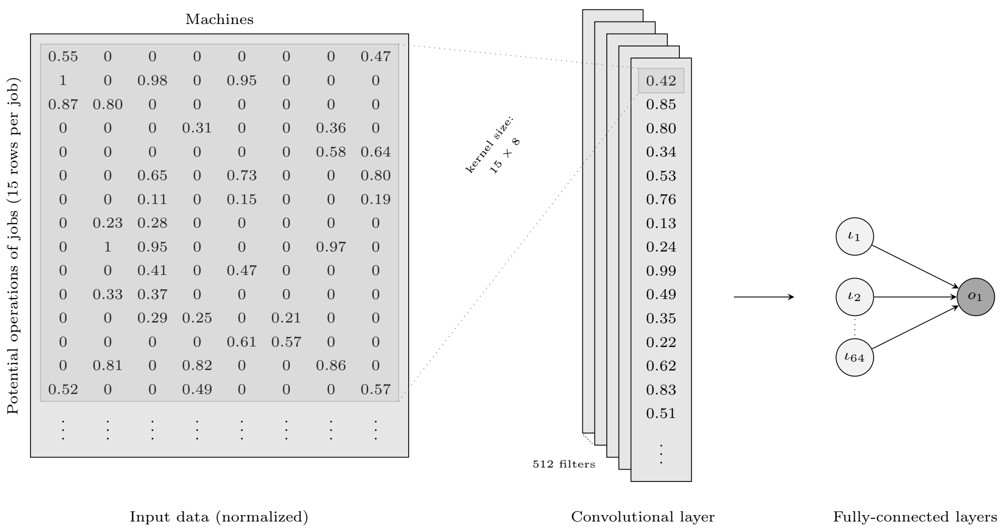


Figure 5: Illustration of the convolutional neural network

are represented by a 600 × 8 matrix (a maximum of 40 jobs with a maximum of 15 operations on a maximum of 8 machines) with 4800 elements. A zero within this matrix indicates that the corresponding operation cannot be processed on the respective machine (hence, it does not exist if all elements within a row are zero), while a one is associated to the largest possible processing time value (in our case 110). As before, the learning process was performed with Adam (learning rate 0.005, all other parameters with default values, 5 epochs, batch size 100). Based on an empirical evaluation on the validation set, we decided to make use of the following structure.

In a convolutional layer, the information included in this matrix is aggregated using a kernel size of 15 $\times$ 8 (a maximum of 15 operations per job on a maximum of 8 machines) and a stride of 15, in order to aggregate the information for each job separately. We use a total of 512 filters. The resulting output is passed through fully connected layers consisting of a 64 units layer with relu activation functions and an output layer composed of one unit with a sigmoid activation function. On the validation set, CNN has an accuracy of 84.1%.

Finally, in order to be able to evaluate the effect of the convolutional layer, we trained an additional fully connected neural network (Adam setup as in case of CNN), denoted by FCNN', that uses the above processing time matrix as input data but does not feature a convolutional layer. As in case of FCNN, it has three hidden layers with 256, 128, and 64 units with relu activation functions. The input layer is adjusted according to the modified input data, while the output layer is identical to the one in FCNN. FCNN' has an accuracy of 82.79% on the validation set. At first glance, this indicates that the use of a convolutional layer pays off when using unprocessed instance data. A detailed analysis follows in Section 5.

4.4.3. Overview of resulting models

Before finally validating our algorithm selection models on a randomly constructed test set, we summarize all of our models in Table 8. It includes the computational times needed for building/training the models. The computation of the feature set of an instance as well as the selection of a corresponding solver by the models takes only a few milliseconds, so that it is of minor interest.

Table 8: Overview of algorithm selection models

Abbreviation	Method	Model	Input data	Build/Training time [s]
LMT	Decision Tree	Logistic model tree	Features	99.26
RF	Decision Tree	Random forest	Features	16.64
REPT	Decision Tree	REP tree	Features	4.37
FCNN	Neural network	Fully connected neural network	Features	143.32
FCNN'	Neural network	Fully connected neural network	Proc. time parameters	220.57
CNN	Neural network	Convolutional neural network	Proc. time parameters	271.81

5. Computational results and evaluation

For the final validation of our algorithm selection models, we generated a separate test set. The generation procedure and size of the instances are in line with Section 4.2. In order to generate previously unseen instances, we used new random seeds. For each reasonable combination of parameter values (see Table 6), we generated 5 test instances. For the resulting 5440 instances, we proceeded as described in Section 4.3, i.e., we called both solvers, CPLEX and OR-Tools, with a time limit of 30 (300) seconds and discarded 837 (1290) instances as there was no clear winner. Both solvers returned a feasible solution for all instances and time limits. We denote the remaining 4603 (4150) instances as *test set A* (*test set B*). Note that, even though our algorithm selection models were trained with a time limit of 30 seconds, test set B allows to analyze the performance of our approaches when increasing the time limit for the selected solver to 300 seconds. In addition to these test sets, we additionally validated our algorithm selection models on the 247 literature instances introduced in Section 3.2. For 20 (27) instances no clear winner (CPLEX or OR-Tools) was found for a time limit of 30 (300) seconds. The remaining 227 (220) instances are denoted as *literature set A* (*literature set B*).

As an additional performance indicator, we used *k-fold cross-validation* on the training set (see, e.g., Han et al., 2011) with a time limit of 30 seconds. To do so, we randomly partitioned the training set into k data folds $F_1, \dots, F_k$ with $|F_i| - |F_j| \leq 1$ for all $i, j \in \{1, 2, \dots, k\}$. The training and testing procedure was repeated k times. That is, in k iterations, $i = 1, \dots, k$, the models were trained with the union of all data folds except F_i. Each of the resulting iterations resulted in specific model variants, that were then validated with data fold F_i. For instance, in the first (second) iteration, the union of the data folds $F_2, \dots, F_k$ ($F_1, F_3, \dots, F_k$) was used to train the models, which were then tested on data fold F_1 (F_2). We set $k = 10$. As a performance indicator, we computed the average accuracy over all iterations.

Our overall approach is illustrated in Figure 6.

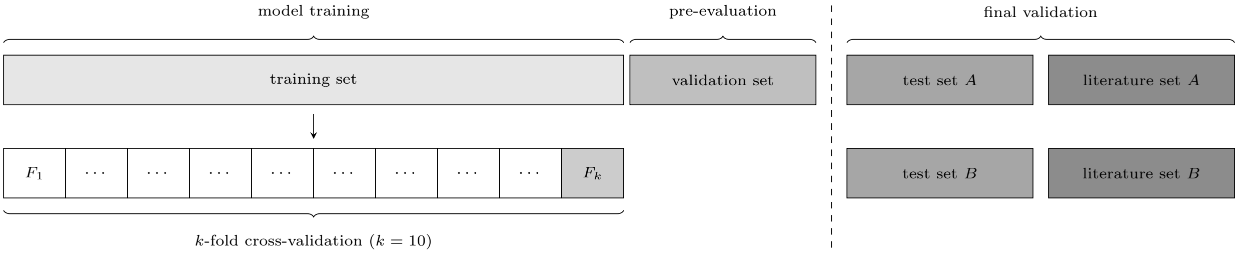


Figure 6: Illustration of training, validation, test and literature sets as well as k-fold cross-validation

5.1. *Evaluation of the algorithm selection models*

The results of the final evaluation on test sets A and B are presented in Tables 9 and 10, respectively. The

Table 9: Final validation on test set A

	CPLEX	OR-Tools	LMT	RF	REPT	FCNN	FCNN'	CNN	VBS
Accuracy [%]	-	-	85.7	86.62	84.55	86.2	82.08	83.01	100
opt. [%]	23.9	28.31	28.22	28.22	28.24	28.24	28.22	28.16	28.33
Q_{avg}	3.11	12.44	2.98	2.97	3.01	2.98	3.04	2.99	2.92
t_{avg} [s]	23.41	22.08	22.06	22.06	22.05	22.06	22.06	22.06	22.02
score	2410	2193	3945	3987	3892	3968	3778	3821	4603

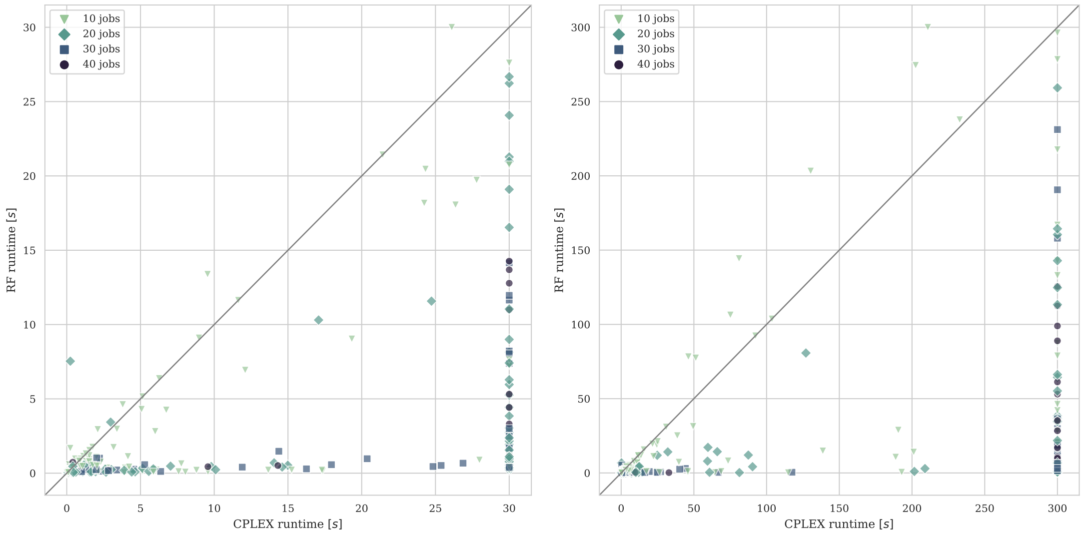
Table 10: Final validation on test set B

	CPLEX	OR-Tools	LMT	RF	REPT	FCNN	FCNN'	CNN	VBS
Accuracy [%]	-	-	74.48	75.04	74.17	76.41	72.58	71.98	100
opt. [%]	29.28	34.82	34.82	34.8	34.8	34.84	34.87	34.84	34.94
Q_{avg}	3.09	4.61	3.01	3	3	2.99	3.01	3.02	2.88
t_{avg} [s]	215.26	200.23	200.17	200.25	200.24	200.17	200.11	200.14	199.88
score	1730	2420	3091	3114	3078	3171	3012	2987	4150

tables illustrate the performance of our algorithm selection models, the use of the individual solvers CPLEX and OR-Tools, as well as the use of the *virtual best solver* (VBS), i.e., the use of an oracle that reveals the best solver for each instance. VBS provides a benchmark on the best possible performance. The tables present information on the percentage of instances for which a provably optimal solution was found with the selected solver within the given time limit (row "opt."), the average quality ratios (row "Q_{avg}"), the average runtimes (row "t_{avg}"), and the sum of scoring points (row "score") over all instances. Moreover, they include information on the accuracy of the selectors (row "Accuracy"). Again, the time needed to compute the feature values of an instance is negligible.

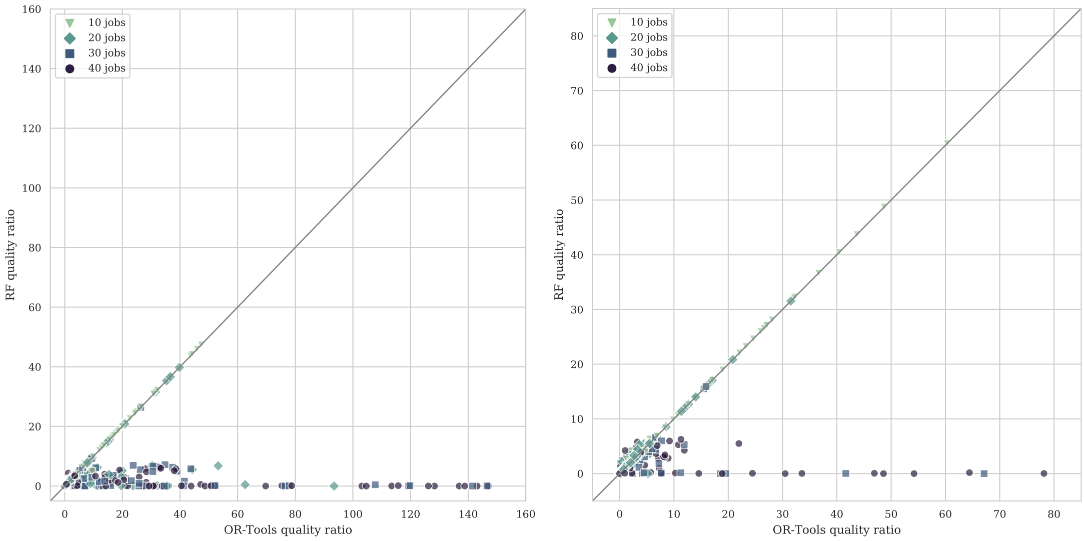
We observe that all of our algorithm selection models perform similar or better than the individual solvers with respect to all performance indicators for both test sets. As to be expected, the positive effect of using our selectors is more significant for test set A. The accuracy values are in similar ranges for all models but FCNN', that suffers from the absence of the convolutional layer when compared with CNN. RF and FCNN tend to provide the best overall performance on the test sets A and B, respectively. Moreover, the selectors that make use of the pre-calculated features outperform the other selectors. However, this effect is rather small when a convolutional layer is used (CNN). Hence, in practice, planners can abstain from the selection and computation of these features at little cost.

As already observed in Sections 3 and 4, CPLEX and OR-Tools show a performance complementary. For the test set, CPLEX receives 2410 (1730) and OR-Tools receives 2193 (2420) scoring points for test set A (B). Again, CPLEX tends to provide better quality ratios while OR-Tools wins when considering the ability to quickly determine optimal solutions. As exemplarily illustrated for RF in Figures 7 and 8, these effects are leveraged by our algorithm selection approaches. The scatter plots presented in these figures are constructed as the ones



(a) Test set A (b) Test set B

Figure 7: Comparison of CPLEX and RF for instances for which at least one approach resulted in an optimal solution



(a) Test set A (b) Test set B

Figure 8: Comparison of quality ratios when using OR-Tools and RF

in Section 3.3. Figure 7 illustrates the runtimes on the instances for which CPLEX or the solver selected by RF was able to determine an optimal solution. Figure 8 presents the quality ratios of the solutions returned by OR-Tools and the solver selected by RF. For the sake of clarity, the plots focus on a representative subset of instances. We observe that RF outperforms CPLEX with respect to the runtimes needed to compute optimal solutions. Additionally, the use of RF results in better quality ratios than the individual use of OR-Tools for the majority of test instances. Hence, RF incorporates the performance benefits of both solvers. This effect can be observed for both test sets, while it is larger on test set A.

The performance of our algorithm selection models on literature sets A and B is presented in Tables 11 and 12, respectively. Due to the different characteristics of the literature instances when compared with the instances of the training set (up to 18 machines and 25 operations per job), we adjusted the processing time matrix for FCNN' and CNN, i.e., the input data is represented by a 1000 $\times$ 18 matrix (a maximum of 40 jobs with a maximum of 25 operations on a maximum of 18 machines).

Table 11: Validation on literature set A

	CPLEX	OR-Tools	LMT	RF	REPT	FCNN	FCNN'	CNN	VBS
Accuracy [%]	-	-	47.58	71.37	62.56	66.08	68.72	77.09	100
opt. [%]	55.95	54.87	55.95	55.95	55.75	55.95	55.75	55.95	55.95
Q_{avg}	13.1	19.15	18.64	13.08	13.21	13.07	13.17	13.15	13.01
t_{avg} $[s]$	14.91	15.57	15.41	14.88	14.97	14.95	14.88	14.81	14.47
score	130	97	108	162	142	150	156	175	227

Table 12: Validation on literature set B

	CPLEX	OR-Tools	LMT	RF	REPT	FCNN	FCNN'	CNN	VBS
Accuracy [%]	-	-	48.64	70	62.27	66.36	66.82	72.27	100
opt. [%]	63.18	63.47	63.18	63.18	63.01	63.64	63.01	63.64	63.64
Q_{avg}	12.94	13.69	13.6	12.91	12.97	12.89	12.99	12.97	12.84
t_{avg} $[s]$	117.22	118.5	117.83	116.35	117.25	115.91	117.46	116.96	114.49
score	127	93	107	154	137	146	147	159	220

With the exception of LMT, we observe that all of our algorithm selection models perform quite well when compared with the use of the individual solvers CPLEX or OR-Tools. With respect to the accuracy and the scoring points, CNN tends to provide the best results. When looking at the percentage of instances for which a provably optimal solution was found or the average quality ratio, RF and FCNN are the overall best choice. As can be seen based on a comparison of the performance of CPLEX with VBS, the benefit of using an our algorithm selection approach for the literature instances will necessarily be rather limited. Nevertheless, our algorithm selection models, in particular RF, FCNN, and CNN, tend to provide slightly better results than CPLEX.

The average accuracy values resulting from the 10-fold cross-validation on the training set are presented in Table 13. We find that the average accuracy values are in similar ranges for all models, with RF being the overall best choice.

Table 13: Results for 10-fold cross-validation on the training set

	LMT	RF	REPT	FCNN	FCNN'	CNN
Avg. accuracy [%]	86.1	87.55	85.26	86.55	83.1	83.54

5.2. Feature subset selection

In Section 4.1, we introduced 151 features that were used to train the models. In order to detect a subset of the most relevant features and allow shorter training times, we applied a correlation-based feature subset selection procedure (see Hall, 1998) using a best first search strategy with a bi-directional search provided by WEKA. The procedure was performed on the training set. It selected four instance size related features, five flexibility related features, one processing time related feature, and seven priority rule related features. These 17 features were then used for training the three selected decision tree variants and the fully connected neural network as illustrated in Sections 4.4.1 and 4.4.2. The resulting models are referred to with an additional index $\circ$: LMT$^\circ$, RF$^\circ$, REPT$^\circ$ and FCNN$^\circ$. The performance of these algorithm selection models on the test sets is presented in Table 14. Values in parentheses illustrate the difference of the corresponding values compared to the values in Tables 9 and 10 in order to better evaluate the effect of the feature subset selection. We observe

Table 14: Performance of the selectors when trained with the reduced feature set

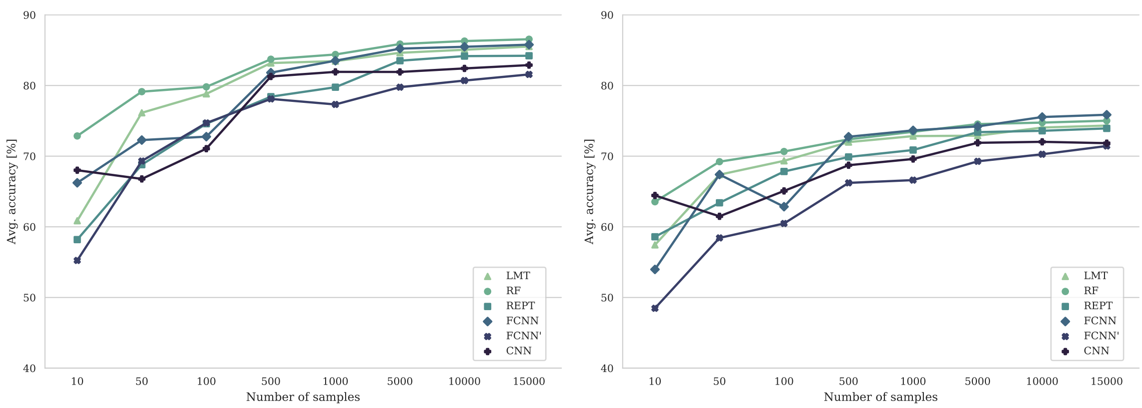
	Test set A				Test set B			
	LMT$^\circ$	RF$^\circ$	REPT$^\circ$	FCNN$^\circ$	LMT$^\circ$	RF$^\circ$	REPT$^\circ$	FCNN$^\circ$
Accuracy [%]	85.55 (−0.15)	86.14 (−0.48)	85.07 (+0.52)	85.27 (−0.93)	74.24 (−0.24)	74.92 (−0.12)	74.75 (+0.58)	75.30 (−1.11)
opt. [%]	28.18 (−0.04)	28.24 (−0.02)	28.2 (−0.04)	28.24	34.84 (+0.02)	34.84 (+0.04)	34.75 (−0.05)	34.82 (−0.02)
Q_{avg}	2.97 (−0.01)	2.98 (+0.01)	2.99 (−0.02)	2.98	3 (−0.01)	3	3	3 (+0.01)
t_{avg} [s]	22.06	22.05 (−0.01)	22.08 (+0.03)	22.06	200.18 (+0.01)	200.1 (−0.15)	200.31 (+0.07)	200.17
score	3938 (−7)	3965 (−22)	3916 (+24)	3925 (−43)	3081 (−10)	3109 (−5)	3102 (+24)	3125 (−46)

that the use of the reduced feature set does not significantly impair the performance of the algorithm selection models. In case of REPT, there are even slight improvements.

5.3. Reducing the size of the training set

In this section, we analyze the impact of a reduced training set size on the performance of our selectors. This is of major importance for decision makers in practice, as it specifies the number of samples needed in order to obtain adequate results.

We randomly selected subsets of instances of the training set with 10, 50, 100, 500, 1000, 5000, 10000, and 15000 instances, respectively. These reduced training sets were then used to train our models. This was done 10 times for each combination of training set size and model. The average performance of the resulting selectors on test sets A and B is presented in Figure 9. The plots show the average accuracy of the different selectors over



(a) Test set A (b) Test set B

Figure 9: Performance of algorithm selection approaches when using sparse training data

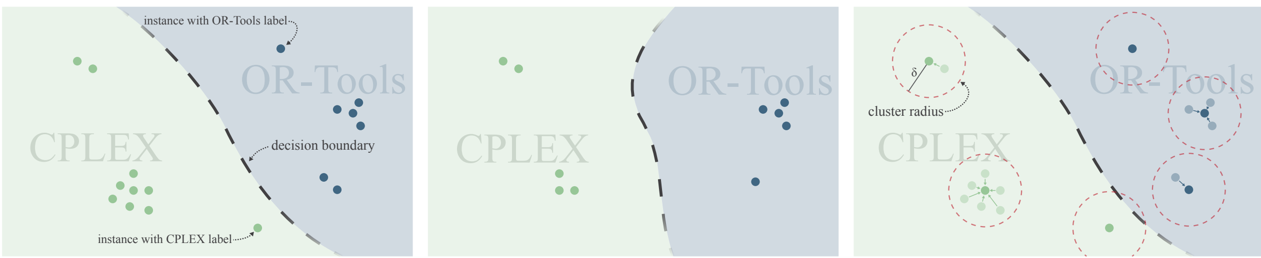
the size of the training set that was used to train the models. For the sake of readability, the sample size values are arranged equidistantly on the abscissa.

As can be seen, while an increasing sample size has a positive effect on the average performance of the selectors, most selectors perform quite well even for very small sample sizes. The positive effect of increasing the sample size tends to diminish when exceeding an amount of 1000 instances. Overall, RF tends to perform best across the varying samples sizes. Hence, in practice, varying characteristics as well as previously unknown characteristics of representative instances, e.g., due to the availability of new machines or the production of new products, can be successfully incorporated into the selectors based on only a few data samples. As only a relatively small amount of data samples is needed, the time required for the collection or generation of these new data samples is significantly smaller when compared with the time needed in case of a large data set. This can be a decisive factor for the application of algorithm selectors in real-world manufacturing systems. It can be concluded that a large number of samples is not necessarily needed, but that it is more important to make use of a set of samples that includes all relevant problem characteristics.

5.4. Reducing the training set to relevant instances

The previous section has demonstrated that it is possible to train the selectors with relatively few instances. However, in some cases, larger training sets are available or it may simply be beneficial to take account of larger training sets. On the other hand, when tuning parameters of deep neural networks, the time needed for evaluating different networks can be reduced when using smaller training sets, i.e., the corresponding training times can be reduced when using less instances. In this section, we therefore analyze the question of whether datasets can be reduced more effectively than by applying pure random subsampling (see Figure 10). We restrict our attention to FCNN and CNN. When subsampling randomly, chances are high that isolated samples are deleted even though they may be relevant for the training process as they have a major impact on the final model and its prediction accuracy. Reducing the training set in a strategic way, on the other hand, may allow to effectively train and evaluate more models.

In order to reduce the size of the training set in a more strategic manner, we developed a simple procedure that is based on dividing the training set into two subsets based on the binary instance labels (indicating the best solvers for the instances, see Figure 10a) and then detecting clusters of similar instances within these subsets. To



(a) Instance subsets based on labels (b) Random subsampling (c) Instance clusters

Figure 10: Exemplary illustration the construction of a reduced training set

do so, each instance is associated to a point in $\mathbb{R}^5$ that corresponds to the normalized values of the following five features (see Section 4.1): number of jobs, number of machines, arithmetic mean of the number of operations of the jobs, arithmetic mean of the number of eligible machines for each operation, corrected standard deviation of the corrected standard deviation of the average processing times of the operations. These features reflect the attributes used for generating the instances (see Section 4.2). In order to measure the degree of similarity of two instances, we compute the Euclidian distance between the corresponding points. Two instances are defined to be similar, if this distance is not larger than a given threshold δ.

We proceed as follows. First, we construct an ordering of the instances of each of the two subsets of the training set (see Figure 10a). To do so, we consider the first of the aforementioned features, i.e., the number of jobs, and order the instances in non-decreasing order of the respective feature values. The other four aforementioned features serve as tie-breakers (in the above sequence) when all previously considered feature values of two instances are identical. We then construct clusters of similar instances as follows. For each ordering, we initialize a first cluster with the first instance of the ordering and add all similar instances of the ordering, i.e., all instances of the corresponding instance subset that lie within a hypersphere of radius δ around this instance (see Figure 10c). All instances that have been clustered are then deleted from the respective ordering and the process repeats until all instances are associated to a cluster. We then randomly select exactly one instance of each cluster to become an element of the reduced training set. It is associated to a weight that is set to the number of instances that are included in the cluster divided by the total number of instances in the original training set. As a result, larger values of δ result in fewer samples in the reduced training set. This process can be interpreted as randomly merging similar instances to a single sample (see Figure 10c). The weights compensate for this reduction. Hence, isolated samples are not deleted from the training set as it is potentially done in case of random subsampling (see Figure 10b). Instead, isolated samples are taken account of with a small weight. The loss function (1) is adapted accordingly:

$$L' = -\frac{1}{|K|}\sum_{k=1}^{|K|}(1+\alpha\lambda_k)[y_k \log(\tilde{y}_k) + (1-y_k)\log(1-\tilde{y}_k)] \tag{2}$$

Here, λ_k represents the weight of instance k in the reduced training set and α is an additional parameter to regularize the impact of this balancing factor.

We analyzed various parameter settings, $\delta \in 0.5, 0.4, 0.2$, for constructing the instance clusters. This resulted in a total of 62 ($\delta = 0.5$), 124 ($\delta = 0.4$) and 795 ($\delta = 0.2$) samples. As the selection of the specific instances is random, we generated ten different reduced training sets for each threshold value. Then, we trained the models FCNN and CNN on the resulting training sets and determined the accuracy with respect to test set A. The results are presented in Table 15. It presents information about the average accuracy and the corrected standard

Table 15: Performance using a weighting sampling strategy

	RF		FCNN		CNN	
	Avg. accuracy [%]	Std. deviation	Avg. accuracy [%]	Std. deviation	Avg. accuracy [%]	Std. deviation
62 samples (weighted, $\delta = 0.5$)						
$\alpha = 0$	-	-	58.66	6.51	64.67	12.08
$\alpha = 100$	-	-	74.47	4.43	72.57	6.97
$\alpha = 200$	-	-	74.4	3.39	74.8	6.62
$\alpha = 300$	-	-	75.25	3.6	75.3	6.11
62 samples (random)	79.86	0.88	72.65	4.34	70.11	10.77
124 samples (weighted, $\delta = 0.4$)						
$\alpha = 0$	-	-	73.2	6.87	70.86	7.88
$\alpha = 100$	-	-	78.59	1.47	78.15	2.56
$\alpha = 200$	-	-	78.83	1.33	77.79	2.78
$\alpha = 300$	-	-	78.31	1.51	75.51	2.98
124 samples (random)	80.21	1.52	78.15	2.34	79.63	1.66
795 samples (weighted, $\delta = 0.2$)						
$\alpha = 0$	-	-	81.09	1.25	79.91	0.81
$\alpha = 100$	-	-	81.18	1.38	79.94	1
$\alpha = 200$	-	-	82.02	1.11	80.05	0.77
$\alpha = 300$	-	-	81.87	1.19	80.26	1.07
795 samples (random)	84.29	0.36	82.21	1.28	81.35	0.63

deviation of the accuracy over the ten runs. For comparison, we include results based on random instance subsets of the same size (again, for 10 runs). Furthermore, we evaluate RF on random instance subsets. Note that, because our sampling strategy makes use of the adapted loss function, it is only applicable for the deep neural networks. With respect to the parameter α, we used values $\alpha \in 0, 100, 200, 300$. For $\alpha = 0$, the balancing factors of the corresponding samples are not considered (see Equation (2)).

We observe that our sampling strategy tends to outperform random subsampling for the neural networks. This effect is more pronounced when the number of samples is small, because random subsampling is less likely to delete relevant isolated samples for larger sizes of the reduced training sets. When ignoring the weighting parameters (case $\alpha = 0$), our sampling strategy fails for larger values of δ, as it does not take account of the actual distribution of the instances. Furthermore, in the considered range, our sampling strategy benefits from increasing values of α. However, as already observed in Section 5.3, RF tends to perform best across the varying samples sizes.

With respect to the training times, the computational time needed to train FCNN (CNN) on the smallest subsets consisting of 62 samples is, on average, 133.34 (99.77) seconds. Especially for the case of CNN, this is a significant reduction when compared to the values for the original training set given in Table 8. Hence, our reduction method allows to effectively train and evaluate more models when needed, in particular when setting up deep neural networks.

6. Conclusion

In this paper, we evaluated the performance of different CP solvers for the FJSP on test instances taken from the literature as well as randomly generated test instances. In a computational study, we introduced a scoring system that captures different performance measures (solution quality/optimality, runtime). When solely considering solution quality, the CP solver provided by CPLEX is the clear winner and outperforms the other solvers. However, when additionally considering the ability to quickly compute optimal solutions (as specified in our scoring system), the overall picture changes, and the CP solver provided by OR-Tools, an open source software suite developed by Google, becomes competitive. We leveraged this performance complementarity by developing algorithm selection approaches that aim to select the best of the two solvers for a given instance. These selectors make use of two machine learning techniques, decision trees and deep neural networks, and are restricted to this binary classification. Unfortunately, our performance evaluation has shown that alternative non-commercial solvers are currently not competitive when compared with the CP solvers provided by CPLEX and Google. Hence, when relying on standard solvers, which is usually very attractive for practitioners, there is currently no alternative to these binary decisions. However, our algorithm selection approaches can easily be adapted to multiclass classification once other solvers become a relevant alternative.

We trained and validated the resulting algorithm selection approaches on a large set of random test instances. In an extensive computational study, we demonstrated that our models outperform the use of a single solver when restricting the runtimes to relatively small values. In general, all of our selectors perform similarly well. On average, a decision tree technique based on random forests tends to provide the best performance. The building and training process of the selectors solely has to be performed one single time and is not relevant for the everyday planning process. Hence, as the computation of the feature values of an instance as well as the selection of a corresponding solver by the models only takes a few milliseconds, our approaches are well suited for real-time scheduling settings in practice, where it is of major importance to quickly compute high quality feasible schedules in order to guarantee smooth production processes without unnecessary disruptions. This allows to conclude that our approaches should be considered as a relevant tool by decision makers in practice.

The finding that all of our algorithm selection models perform similarly well is independent of the use of decision trees or deep neural networks. When comparing these two machine learning techniques, the configuration of deep neural networks is a more challenging task, since it requires extensive parameter tuning. On the other hand, deep neural networks are well suited for constructing selectors that do not rely on the computation of feature values but work with unprocessed instance data. When relying on features, selection strategies can be used to determine features that are most relevant. We observed that the use of a reduced feature subset for the training process does not significantly impair the performance of the algorithm selection models.

When wanting to apply algorithm selection models to problem instances that significantly differ from the ones of the training set, the corresponding models should be trained again in order to achieve good results. In industrial settings, this can, for example, become relevant when restructuring production processes, when introducing new products, when buying additional machines or when modernizing the current machines. In this context, we observed that new instance characteristics can be successfully incorporated into our algorithm selection models based on only a few data samples. Finally, we developed a method to reduce the size of the training set to a subset of the most relevant instances. In practice, this method allows to effectively train and evaluate more models.

In the context of deep neural networks, it has to be noted that their interpretability is a challenging task. For future research, it would be beneficial to propose methods to better understand and interpret the behavior of deep neural networks. In case of algorithm selection approaches, knowledge about the relative strengths and weaknesses of different algorithms could be helpful for improving algorithmic details of specific solution approaches. Future research may also target at the construction of algorithm selection approaches for practically relevant extensions of the FJSP that, for example, integrate the assignment of machine or setup operators or that take account of blocking or no-wait constraints. For these settings, out-of-the-box algorithms may not be as easy to use as in case of the classical FJSP. Hence, this research may have to include the development of algorithms that show a complementary performance on instance sets with different features. Moreover, while we have focussed on computing feasible solutions within a given time limit, it will also be interesting to consider algorithm selection approaches that aim at determining optimal solutions or that integrate decisions on reasonable time limits.

References

Adams, J., Balas, E., & Zawack, D. (1988). The shifting bottleneck procedure for job shop scheduling. *Management Science*, *34*, 391–401.

Applegate, D., & Cook, W. (1991). A computational study of the job-shop scheduling problem. *ORSA Journal on Computing*, *3*, 149–156.

Behnke, D., & Geiger, M. J. (2012). *Test instances for the flexible job shop scheduling problem with work centers*. Technical Report RR-12-01-01 Helmut Schmidt University Hamburg.

Bengio, Y., Lamblin, P., Popovici, D., & Larochelle, H. (2007). Greedy layer-wise training of deep networks. In B. Schölkopf, J. C. Platt, & T. Hoffman (Eds.), *Proceeding of the 19th Conference on Neural Information Processing Systems* (pp. 153–160). Cambridge: MIT Press.

Bengio, Y., Lodi, A., & Prouvost, A. (2021). Machine learning for combinatorial optimization: a methodological tour d'horizon. *European Journal of Operational Research*, *290*, 405–421.

Błażewicz, J., Domschke, W., & Pesch, E. (1996). The job shop scheduling problem: Conventional and new solution techniques. *European Journal of Operational Research*, *93*, 1–33.

Blazewicz, J., Ecker, K. H., Pesch, E., Schmidt, G., Sterna, M., & Weglarz, J. (2019). *Handbook on Scheduling: From Theory to Practice*. (2nd ed.). Berlin: Springer.

Boyes, H., Hallaq, B., Cunningham, J., & Watson, T. (2018). The industrial internet of things (IIoT): an analysis framework. *Computers in Industry*, *101*, 1–12.

Brandimarte, P. (1993). Routing and scheduling in a flexible job shop by tabu search. *Annals of Operations Research*, *41*, 157–183.

Breiman, L. (2001). Random forests. *Machine Learning*, *45*, 5–32.

Brucker, P., & Schlie, R. (1990). Job-shop scheduling with multi-purpose machines. *Computing*, *45*, 369–375.

Burdett, R. L., Corry, P., Eustace, C., & Smith, S. (2020). A flexible job shop scheduling approach with operators for coal export terminals – a mature approach. *Computers & Operations Research*, *115*, 104834.

Burke, E. K., Gendreau, M., Hyde, M., Kendall, G., Ochoa, G., Özcan, E., & Qu, R. (2013). Hyper-heuristics: a survey of the state of the art. *Journal of the Operational Research Society*, *64*, 1695–1724.

Carlier, J., & Pinson, E. (1989). An algorithm for solving the job-shop problem. *Management Science*, *35*, 164–176.

Cenamor, I., De La Rosa, T., & Fernández, F. (2016). The IBaCoP planning system: instance-based configured portfolios. *Journal of Artificial Intelligence Research*, *56*, 657–691.

Chambers, J. B., & Barnes, J. W. (1996). *Flexible job shop scheduling by tabu search*. Technical Report ORP96-09 Graduate Program in Operations and Industrial Engineering, The University of Texas at Austin.

Chaudhry, I. A., & Khan, A. A. (2016). A research survey: review of flexible job shop scheduling techniques. *International Transactions in Operational Research*, *23*, 551–591.

Chu, G., Stuckey, P. J., Schutt, A., Ehlers, T., Gange, G., & Francis, K. (2019). Chuffed solver documentation. `https://github.com/chuffed/chuffed`. Last accessed 2020-11-18.

Da Col, G., & Teppan, E. C. (2019a). Google vs IBM: a constraint solving challenge on the job-shop scheduling problem. arXiv preprint arXiv:1909.08247.

Da Col, G., & Teppan, E. C. (2019b). Industrial size job shop scheduling tackled by present day CP solvers. In T. Schiex, & S. de Givry (Eds.), *Proceedings of the 25th International Conference on Principles and Practice of Constraint Programming* (pp. 144–160). Cham: Springer.

Dauzère-Pérès, S., & Paulli, J. (1997). An integrated approach for modeling and solving the general multiprocessor job-shop scheduling problem using tabu search. *Annals of Operations Research*, *70*, 281–306.

Dorndorf, U., & Pesch, E. (1995). Evolution based learning in a job shop scheduling environment. *Computers & Operations Research*, *22*, 25–40.

Dorndorf, U., Pesch, E., & Phan-Huy, T. (2000). Constraint propagation techniques for disjunctive scheduling problems. *Artificial Intelligence*, *122*, 189–240.

Dorndorf, U., Pesch, E., & Phan-Huy, T. (2002). Constraint propagation and problem decomposition: a preprocessing procedure for the job shop problem. *Annals of Operations Research*, *115*, 125–145.

Fisher, H., & Thompson, G. L. (1963). Probabilistic learning combinations of local job-shop scheduling rules. In J. F. Muth, & G. L. Thompson (Eds.), *Industrial Scheduling* (pp. 225–251). Englewood Cliffs: Prentice-Hall.

Frank, E., Hall, M. A., & Witten, I. H. (2016). *The WEKA Workbench. Online Appendix for "Data Mining: Practical Machine Learning Tools and Techniques"*. (4th ed.). Cambridge: Morgan Kaufmann.

Gershwin, S. B. (2018). The future of manufacturing systems engineering. *International Journal of Production Research*, *56*, 224–237.

Ghaleb, M., Zolfagharinia, H., & Taghipour, S. (2020). Real-time production scheduling in the Industry-4.0 context: addressing uncertainties in job arrivals and machine breakdowns. *Computers & Operations Research*, *123*, 105031.

Giffler, B., & Thompson, G. L. (1960). Algorithms for solving production-scheduling problems. *Operations Research*, *8*, 487–503.

Goodfellow, I., Bengio, Y., & Courville, A. (2016). *Deep Learning*. MIT Press. `http://www.deeplearningbook.org`.

Google (2019a). Google OR-Tools. `https://developers.google.com/optimization/`. Last accessed 2020-11-18.

Google (2019b). Google OR-Tools: Scheduling overview. `https://developers.google.com/optimization/scheduling`. Last accessed 2020-11-18.

Hall, M., Frank, E., Holmes, G., Pfahringer, B., Reutemann, P., & Witten, I. H. (2009). The WEKA data mining software: an update. *SIGKDD Explorations Newsletter*, *11*, 10–18.

Hall, M. A. (1998). *Correlation-based Feature Subset Selection for Machine Learning*. Ph.D. thesis University of Waikato Hamilton, New Zealand.

Ham, A. M., & Cakici, E. (2016). Flexible job shop scheduling problem with parallel batch processing machines: MIP and CP approaches. *Computers & Industrial Engineering*, *102*, 160–165.

Han, J., Kamber, M., & Pei, J. (2011). *Data Mining: Concepts and Techniques*. (3rd ed.). Waltham: Elsevier.

Hart, E., Ross, P., & Nelson, J. (1998). Solving a real-world problem using an evolving heuristically driven schedule builder. *Evolutionary Computation*, *6*, 61–80.

Haupt, R. (1989). A survey of priority rule-based scheduling. *OR Spektrum*, *11*, 3–16.

Hinton, G. E., Osindero, S., & Teh, Y.-W. (2006). A fast learning algorithm for deep belief nets. *Neural Computation*, *18*, 1527–1554.

Hulten, G., Spencer, L., & Domingos, P. (2001). Mining time-changing data streams. In *Proceedings of the 7th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining* (pp. 97–106). New York: ACM.

Hurink, J., Jurisch, B., & Thole, M. (1994). Tabu search for the job-shop scheduling problem with multi-purpose machines. *OR Spektrum*, *15*, 205–215.

Hutter, F., López-Ibáñez, M., Fawcett, C., Lindauer, M., Hoos, H. H., Leyton-Brown, K., & Stützle, T. (2014). AClib: a benchmark library for algorithm configuration. In P. M. Pardalos, M. G. Resende, C. Vogiatzis, & J. L. Walteros (Eds.), *Proceedings of the 8th International Conference on Learning and Intelligent Optimization* (pp. 36–40). Cham: Springer.

Iba, W., & Langley, P. (1992). Induction of one-level decision trees. In D. Sleeman, & P. Edwards (Eds.), *Proceedings of the 9th International Conference on Machine Learning* (pp. 233–240). San Francisco: Morgan Kaufmann.

IBM (2019a). IBM ILOG CPLEX optimization studio 12.9.0: Online documentation. `https://www.ibm.com/support/knowledgecenter/SSSA5P_12.9.0/ilog.odms.studio.help/Optimization_Studio/topics/COS_home.html`. Last accessed 2020-11-18.

IBM (2019b). IBM ILOG CPLEX optimization studio 12.9.0: Scheduling examples. `https://www.ibm.com/support/knowledgecenter/SSSA5P_12.9.0/ilog.odms.ide.help/OPL_Studio/usroplexamples/topics/opl_cp_examples_scheduling.html`. Last accessed 2020-11-18.

Kerschke, P., Hoos, H. H., Neumann, F., & Trautmann, H. (2019). Automated algorithm selection: survey and perspectives. *Evolutionary Computation*, *27*, 3–45.

Kerschke, P., Kotthoff, L., Bossek, J., Hoos, H. H., & Trautmann, H. (2018). Leveraging TSP solver complementarity through machine learning. *Evolutionary Computation*, *26*, 597–620.

Kingma, D. P., & Ba, J. L. (2014). Adam: a method for stochastic optimization. arXiv preprint arXiv:1412.6980.

Kraus, M., Feuerriegel, S., & Oztekin, A. (2020). Deep learning in business analytics and operations research: models, applications and managerial implications. *European Journal of Operational Research*, *281*, 628–641.

Kress, D., & Müller, D. (2019). Mathematical models for a flexible job shop scheduling problem with machine operator constraints. *IFAC-PapersOnLine*, *52*, 94–99.

Kress, D., & Müller, D. (2022). Semiconductor final-test scheduling under setup operator constraints. *Computers & Operations Research*, *138*, 105619.

Kress, D., Müller, D., & Nossack, J. (2019). A worker constrained flexible job shop scheduling problem with sequence-dependent setup times. *OR Spectrum*, *41*, 179–217.

Laborie, P., & Rogerie, J. (2016). Temporal linear relaxation in IBM ILOG CP Optimizer. *Journal of Scheduling*, *19*, 391–400.

Laborie, P., Rogerie, J., Shaw, P., & Vilím, P. (2018). IBM ILOG CP optimizer for scheduling. *Constraints*, *23*, 210–250.

Landwehr, N., Hall, M., & Frank, E. (2005). Logistic model trees. *Machine Learning*, *59*, 161–205.

Lawrence, S. (1984). *Resouce constrained project scheduling: an experimental investigation of heuristic scheduling techniques*. Technical Report Graduate School of Industrial Administration, Carnegie-Mellon University.

LeCun, Y., Kavukcuoglu, K., & Farabet, C. (2010). Convolutional networks and applications in vision. In *Proceedings of the IEEE International Symposium on Circuits and Systems* (pp. 253–256). IEEE.

Lenstra, J. K., & Rinnooy Kan, A. H. G. (1979). Computational complexity of discrete optimization problems. *Annals of Discrete Mathematics*, *4*, 121–140.

Lunardi, W. T., Birgin, E. G., Laborie, P., Ronconi, D. P., & Voos, H. (2020). Mixed integer linear programming and constraint programming models for the online printing shop scheduling problem. *Computers & Operations Research*, *123*, 105020.

Lunardi, W. T., Birgin, E. G., Ronconi, D. P., & Voos, H. (2021). Metaheuristics for the online printing shop scheduling problem. *European Journal of Operational Research*, *293*, 419–441.

Mastrolilli, M. (2020). Flexible job shop problem. `http://people.idsia.ch/~monaldo/fjsp.html`. Last accessed 2020-12-08.

Mastrolilli, M., & Gambardella, L. M. (2000). Effective neighbourhood functions for the flexible job shop problem. *Journal of Scheduling*, *3*, 3–20.

McCulloch, W. S., & Pitts, W. (1943). A logical calculus of the ideas immanent in nervous activity. *The Bulletin of Mathematical Biophysics*, *5*, 115–133.

Messelis, T., & De Causmaecker, P. (2014). An automatic algorithm selection approach for the multi-mode resource-constrained project scheduling problem. *European Journal of Operational Research*, *233*, 511–528.

MiniZinc (2013). MiniZinc challenge 2013. `https://www.minizinc.org/challenge2013/challenge.html`. Last accessed 2020-11-18.

MiniZinc (2020). MiniZinc challenge 2020. `https://www.minizinc.org/challenge2020/challenge.html`. Last accessed 2020-11-18.

Mirshekarian, S., & Šormaz, D. N. (2016). Correlation of job-shop scheduling problem features with scheduling efficiency. *Expert Systems with Applications*, *62*, 131–147.

Müller, D., & Kress, D. (2021). Filter-and-fan approaches for scheduling flexible job shops under workforce constraints. *International Journal of Production Research*, *in press*. doi:`https://doi.org/10.1080/00207543.2021.1937745`.

Müller, D., Müller, M. G., Kress, D., & Pesch, E. (2021). Test instances for the FJSP. `https://doi.org/10.6084/m9.figshare.13522625.v1`. Last accessed 2021-01-14.

Nethercote, N., Stuckey, P. J., Becket, R., Brand, S., Duck, G. J., & Tack, G. (2007). MiniZinc: towards a standard CP modelling language. In C. Bessière (Ed.), *Proceedings of the 13th International Conference on Principles and Practice of Constraint Programming* (pp. 529–543). Berlin: Springer.

Pesch, E., & Tetzlaff, U. A. W. (1996). Constraint propagation based scheduling of job shops. *INFORMS Journal on Computing*, *8*, 144–157.

Prud'homme, C., Fages, J.-G., & Lorca, X. (2016). *Choco documentation*. TASC, INRIA Rennes, LINA CNRS UMR 6241, COSLING S.A.S. `https://choco-solver.org`. Last accessed 2020-11-18.

Pulina, L., & Tacchella, A. (2009). A self-adaptive multi-engine solver for quantified Boolean formulas. *Constraints*, *14*, 80–116.

PyTorch (2020). PyTorch online documentation. `https://pytorch.org/docs/stable/index.html`. Last accessed 2020-11-18.

Quinlan, J. R. (1993). *C4.5: Programs for Machine Learning*. Morgan Kaufmann.

Rakovitis, N., Li, D., Zhang, N., Li, J., Zhang, L., & Xiao, X. (2022). Novel approach to energy-efficient flexible job-shop scheduling problems. *Energy*, *238*, 121773.

Rice, J. R. (1976). The algorithm selection problem. *Advances in Computers*, *15*, 65–118.

Rizzini, M., Fawcett, C., Vallati, M., Gerevini, A. E., & Hoos, H. H. (2017). Static and dynamic portfolio methods for optimal planning: an empirical analysis. *International Journal on Artificial Intelligence Tools*, *26*, 1760006.

Schmidhuber, J. (2015). Deep learning in neural networks: an overview. *Neural Networks*, *61*, 85–117.

Schulte, C., Tack, G., & Lagerkvist, M. Z. (2019). Modeling and programming with Gecode. `https://www.gecode.org/doc-latest/MPG.pdf`. Last accessed 2020-11-18.

Smith-Miles, K. A. (2008). Towards insightful algorithm selection for optimisation using meta-learning concepts. In *Proceedings of the IEEE International Joint Conference on Neural Networks* (pp. 4118–4124). IEEE.

Smith-Miles, K. A. (2009). Cross-disciplinary perspectives on meta-learning for algorithm selection. *ACM Computing Surveys*, *41*, 1–25.

Stuckey, P. J., Feydy, T., Schutt, A., Tack, G., & Fischer, J. (2014). The MiniZinc challenge 2008–2013. *AI Magazine*, *35*, 55–60.

Stuckey, P. J., Marriott, K., & Tack, G. (2020). The MiniZinc handbook. `https://www.minizinc.org/doc-2.5.5/en/index.html`. Last accessed 2021-09-03.

Sumner, M., Frank, E., & Hall, M. (2005). Speeding up logistic model tree induction. In A. M. Jorge, L. Torgo, P. Brazdil, R. Camacho, & J. Gama (Eds.), *Proceedings of the 9th European Conference on Principles and Practice of Knowledge Discovery in Databases* (pp. 675–683). Berlin: Springer.

Thenarasu, M., Rameshkumar, K., Rousseau, J., & Anbuudayasankar, S. P. (2022). Development and analysis of priority decision rules using MCDM approach for a flexible job shop scheduling: A simulation study. *Simulation Modelling Practice and Theory*, *114*, 102416.

Türkyılmaz, A., Şenvar, Ö., Ünal, İ., & Bulkan, S. (2022). A hybrid genetic algorithm based on a two-level hypervolume contribution measure selection strategy for bi-objective flexible job shop problem. *Computers & Operations Research*, *in press*, 105694. doi:`https://doi.org/10.1016/j.cor.2021.105694`.

Vázquez-Rodríguez, J. A., & Petrovic, S. (2010). A new dispatching rule based genetic algorithm for the multi-objective job shop problem. *Journal of Heuristics*, *16*, 771–793.

Wagner, M., Lindauer, M., Mısır, M., Nallaperuma, S., & Hutter, F. (2018). A case study of algorithm selection for the traveling thief problem. *Journal of Heuristics*, *24*, 295–320.

Wang, L., Pan, Z., & Wang, J. (2021). A review of reinforcement learning based intelligent optimization for manufacturing scheduling. *Complex System Modeling and Simulation*, *1*, 257–270.

Wari, E., & Zhu, W. (2019). A constraint programming model for food processing industry: a case for an ice cream processing facility. *International Journal of Production Research*, *57*, 6648–6664.

Xu, L., Hutter, F., Hoos, H. H., & Leyton-Brown, K. (2008). SATzilla: portfolio-based algorithm selection for SAT. *Journal of Artificial Intelligence Research*, *32*, 565–606.

Xu, L., Hutter, F., Hoos, H. H., & Leyton-Brown, K. (2012). Evaluating component solver contributions to portfolio-based algorithm selectors. In A. Cimatti, & R. Sebastiani (Eds.), *Proceedings of the 15th International Conference on Theory and Applications of Satisfiability Testing* (pp. 228–241). Berlin: Springer.